CONTINUOUS DYNAMIC OPTIMISATION USING EVOLUTIONARY ALGORITHMS

by

TRUNG THANH NGUYEN

A thesis submitted to
The University of Birmingham
for the degree of
Doctor of Philosophy

School of Computer Science

The University of Birmingham

October 2010

Abstract

Evolutionary dynamic optimisation (EDO), or the study of applying evolutionary algorithms to dynamic optimisation problems (DOPs) is the focus of this thesis.

Based on two comprehensive literature reviews on existing academic EDO research and real-world DOPs, this thesis for the first time identifies some important gaps in current academic research where some common types of problems and problem characteristics have not been covered. In an attempt to close some of these gaps, the thesis makes the following contributions:

First, the thesis helps to characterise DOPs better by providing a new definition framework, two new sets of benchmark problems (for certain classes of continuous DOPs) and several new sets of performance measures (for certain classes of continuous DOPs).

Second, the thesis studies continuous dynamic constrained optimisation problems (DCOPs), an important and common class of DOPs that have not been studied in EDO research. Contributions include developing novel optimisation approaches (with superior results to existing methods), analysing representative characteristics of DCOPs, identifying the strengths/weaknesses of existing methods and suggesting requirements for an algorithm to solve DCOPs effectively.

Third, the thesis studies dynamic time-linkage optimisation problems (DTPs), another important and common class of DOPs that have not been well-studied in EDO research. Contributions include developing a new optimisation approach (with better results than existing methods in certain classes of DTPs), analysing the characteristics of DTPs and the strengths and weaknesses of existing EDO methods in solving certain classes of DTPs.

Acknowledgements

I wish to express my gratitude to my supervisor, Prof. Xin Yao, for his great support and guidance through my study. I thank him for accepting me to be one of his students in the School of Computer Science, University of Birmingham. I am grateful to him for his invaluable advice and his kindness encouragements. I would also like to thank him for encouraging and supporting me to submit my results to conferences or journals whenever possible. I owe special thanks to him for introducing me to the topic of evolutionary dynamic optimisation and to the EADOP project. Although switching to dynamic optimisation after more than six months of pursuing another topic during my PhD study was a difficult decision (to me), I have never regretted. Working in dynamic optimisation gave me the chance to learn more about this challenging and exciting field, and more importantly, gave me the chance to meet and learn from the excellent researchers working in the EADOP project and in the field in general. Without the support of Prof. Yao, I would not be able to achieve what I have done in the thesis.

I would like to thank the School of Computer Science and the Overseas Research Students Awards Scheme for providing me with the financial supports. I also would like to thank the School of Computer Science for the great research environment and facilities they provide.

I especially thank my two examiners: Dr. Jon Rowe and Prof. Yaochu Jin for spending time to examine my thesis and for giving me so many helpful advice and comments. Their advice helped me not only to improve the quality of my thesis, but also equip me with new knowledge and experience to do my research better in the future. I would also like to thank them to make me really enjoyed my viva. Their questions pushed me to the limit, but they also made me feel as comfortable as possible to answer these questions. They also helped me identifying the areas where I can improve my work and the areas that are interesting for future work. I am really grateful to them for that.

I would like to thank the two other members of my Thesis Group, Dr. Jon Rowe and Dr. Ata Kaban for their great questions, suggestions and advice in my Thesis Group meetings to keep me on course. I especially thank Dr. Jon Rowe for his question relating to the normalisation of performance measures, which help me to develop more effective performance measures and for .

I owe special thanks to the excellent researchers that I had the chance to work/discuss with during various EADOP project meetings and AI/NC talks. Specifically, I would like to thank Dr. Philipp Rohlfshagen for his insightful comments during our discussions about dynamic optimisation, especially the ones about the current challenges of evolutionary dynamic optimisation

and for his great works in organising the regular EADOP meetings. He also gave me valuable comments on one of my papers, which form the main parts of Chapter 4 and Chapter 7. I would like to thank Dr. Tabaprata Ray for his advice in our previous discussions, and for his kind helps in classifying in details the fourteen dynamic control problems (and some other problems) among the problems that I reviewed in Chapter 3 and in Table 5 in the Appendix. I would like to thank Prof. Juergen Branke for his insightful comments on my various papers, which result in the works in Chapter 3, 5, 6. I would like to thank Dr. Shengxiang Yang for his comments on my EADOP talks, and for being the best host ever every time we visited the University of Leicester. I would like to thank Prof. Yaochu Jin for giving me valuable advice about dynamic constrained optimisation during our EADOP meetings, and for his email discussions relating to developing benchmark problems. I would like to thank my friend Changhe Li for the discussion and collaborative works we had. Finally, I would like to thank all other researchers with whom I had the chance to work/discuss in the topic of dynamic optimisation: Dr. Hui Cheng, Dr. Hisashi Handa, Lining Xing and Kyriacos Souroullas.

I would also like to thank the many people at the School of Computer Science who have helped me over the time I was pursuing my PhD study, especially those from the Natural Computation Group and CERCIA. I have learnt a lot from their regular AI/NC and Natural Computation talks and from their productive discussions.

I am grateful to Prof. Juergen Branke and Dr. Irene Moser for kindly sending me the source code of their algorithms for study purposes, to Prof. Yaochu Jin, Prof. Beyer and Prof. Suganthan for their fruitful discussions and comments relating to the technical report for the CEC'09 competition on dynamic optimisation, and to the various anonymous reviewers of my related papers for their valuable comments.

I would also like to thank the staffs from the EISU, University of Birmingham for taking a look at some pages of my writing and giving me suggestions to correct my grammatical mistakes.

I also owe my thanks to my friends here in Birmingham and in Vietnam for their supports and encouragements.

Finally, a million of thanks to my family and to Hang, my loving wife. Their unconditional love and sacrifices are the greatest supports for me to complete this long journey. This thesis is dedicated to them.

Contents

List of Figures

List of Tables

List of abbreviations

BOG	Best Of Generation
CH	Constraint Handling
DCOP	Dynamic Constrained Optimisation Problem
DCO	Dynamic Constrained Optimisation
DMO	Dynamic Multi-objective Optimisation
DO	Dynamic Optimisation
DOP	Dynamic Optimisation Problem
DR	Disconnected feasible Region
DTP	Dynamic Time-linkage Problem
dC	dynamic Constraint functions
dF	dynamic objective Function
EA	Evolutionary Algorithm
EC	Evolutionary Computation
EDO	Evolutionary Dynamic Optimisation
fC	fixed Constraint functions
fF	fixed objective Function
GA	Genetic Algorithm
HyperM	Hyper-Mutation GA
MO	Multi-objective Optimisation
(N)NAO	(No) Newly Appearing Optima
(N)RR	(No) Repair Reference population
NU	No Update
noC	no Constraint function
O(N)ICB	Optima (Not) In Constraint Boundary
O(N)ISB	Optima (Not) In Search Boundary
OOR	Out-Of-Range search
RIGA	Random Immigrant GA
(A)RR	(Absolute) Recovery Rate
SwO	Switched optimum between disconnected regions
UPC	Update Per Change
UPG	Update Per Generation

Chapter 1

Introduction

Evolutionary dynamic optimisation (EDO), or the study of applying evolutionary algorithms (EA) to dynamic optimisation problems (DOPs), is an active research topic and has increasingly attracted interest from the evolutionary computation (EC) community. The field is relatively young as most of the studies have been made in the last 20 years with the exception of a few early works such as (Fogel *et al.* 1966, Goldberg & Smith 1987) . Due to its relatively young age, the field still has a lot of open areas with open research questions, of which perhaps one of the most important questions is about how well academic EDO research reflects the common characteristics of DOPs and if there are any types of DOPs that have not been covered by current academic research. The main purpose of this thesis is to investigate this important question and to propose solutions to close some of the gaps in this issue.

1.1 Dynamic problems and dynamic optimisation problems

It is necessary to distinguish between *dynamic problems* (also called dynamic environments or time-dependent problems), which are problems that change over time, and *dynamic optimisation problems* (DOPs), which belong to a special class of dynamic problems that are *solved online by an optimisation algorithm as time goes by.* Of these two types of problems, only DOPs are of interest to EDO research. This is because, no matter how the problem changes, from the perspective of an EA or an optimisation algorithm in general, a time-dependent problem is only different from a static problem if it is solved in a dynamic way, i.e. new solutions are produced to react to changes as time goes by. Otherwise, if future changes can be completely integrated into a static objective function, or if a single robust-to-changes solution can be provided, or if only the current static instance of the time-dependent problem is taken into account, then the

problem can be solved using static optimisation techniques and hence is no longer of interest to EDO.

In spite of this difference between dynamic problems and DOPs, in many EDO studies the terms "dynamic problem" and "dynamic optimisation problem" are not distinguished or are used interchangeably. In fact, many EDO studies use the definitions of dynamic problems, especially the formal definitions, to define DOPs. In these studies, DOPs are either defined as a sequence of static problems linked up by some dynamic rules (Weicker 2000, Weicker 2003, Aragon & Esquivel 2004, Rohlfshagen & Yao 2008, Rohlfshagen & Yao 2010) or as a problem that have time-dependent parameters in its mathematical expression (Bäck 1998, Bosman 2007, Woldesenbet & Yen 2009) without mentioning whether the problems are solved online by an optimisation algorithm or not. For example, below is a formal definition (Bosman 2007) for a DOP with the time variable $t \in \mathbb{T} = \left[0, t^{end}\right], t^{end} > 0$:

$$\begin{aligned} &\max\left\{F_\gamma\left(x\left(t\right)\right)\right\} \text{ subject to } C_\gamma\left(x\left(t\right)\right) = feasible \text{ with} \\ &F_\gamma\left(x\left(t\right)\right) = \int_0^{t^{end}} f_{\gamma(t)}\left(x\left(t\right)\right) dt \\ &C_\gamma\left(x\left(t\right)\right) = \begin{cases} feasible & \text{if } \forall t \in \left[0, t^{end}\right] : C_{\gamma(t)}\left(x\left(t\right)\right) = feasible \\ infeasible & \text{otherwise} \end{cases} \end{aligned} \tag{1.1}$$

where f is a function of $x\left(t\right)$ with time-dependent parameters γ and C is the constraint function.

Although definitions like those cited above can be used to effectively represent time-dependent problems, they do not however show whether a time-dependent problem is different from a static problem from the perspective of an optimisation algorithm and hence are not able to distinguish a DOP from the general time-dependent problems.

Some recent EDO studies have taken into account this difference between static optimisation and dynamic optimisation when specifying the scope of the dynamic problems to be studied. Branke (2001*b*) considered time-dependent problems as dynamic only if "the EA has to cope with these dynamics", Morrison (2004) restricted the DOPs studied in his book to those in which "the underlying fitness landscape changes during the operation of the EA", and Bosman (2007) considered DOPs that "must be solved as time goes by" as "the most practical variant of dynamic optimization". Finally, Jin & Branke (2005) considered time-dependent problems "dynamic" only if the dynamics "are to be taken into account in the optimization process". Problems

satisfying this condition are categorised by the authors as "dynamic fitness functions".

The implication of the cited remarks above is that a time-dependent problem is a dynamic optimisation problem (DOP) only if it is solved online by an optimisation algorithm as time goes by. To make it clearer and to distinguish DOPs from other types of time-dependent problems, I propose the following definition for DOPs:

Definition 1.2 (Dynamic optimisation problem) *Given a dynamic problem f_t, an optimisation algorithm G to solve f_t, and a given optimisation period $\left[t^{begin}, t^{end}\right]$, f_t is called a **dynamic optimisation problem** in the period $\left[t^{begin}, t^{end}\right]$ if during $\left[t^{begin}, t^{end}\right]$ the underlying fitness landscape that G uses to represent f_t changes **and** G has to react to this change by providing new optimal solutions.*[1]

From now on, in this thesis we will use the term dynamic optimisation problems (DOPs) to refer to any problems defined by the above definition. A more formal and detailed DOP definition, which takes into account the common characteristics of DOPs, will be provided later in Chapter 4.

The definition above distinguishes DOPs from the other two types of time-dependent problems, which are solved using different approaches as follows:

1. *Time-dependent problems that are solved by static optimisation approaches*: These time-dependent problems are formulated as a static problem by the optimisation algorithm. This is because either all future changes are known and hence can be completely integrated into a static objective function, or only the current static instance of the time-dependent problem is taken into account. In this case the goal is to find a single, static solution using a static optimisation approach

2. *Time-dependent problems that are solved by robust optimisation approaches*: In these problems future changes are normally not completely known but in most cases the problem is still formulated as a static problem with an expected fitness function. The goal is to find a single solution that is less sensitive under future disturbances such as production tolerances, operating conditions or modelling inaccuracies. A recent review of robust op-

[1] This definition also covers the robust-optimisation-over-time situation described in (Yu *et al.* 2010) where a sequence of $\langle S_1, ...S_k \rangle$ robust solutions is found provided that $k > 1$.

timisation can be found in (Beyer & Sendhoff 2007). Another summary of evolutionary robust optimisation approaches is presented in (Jin & Branke 2005).

1.2 Scope of the thesis

The range of dynamic optimisation problems is large and diversified. In Chapter 3 I will try to cover a wide range of DOPs (including continuous/combinatorial and single/multiple-objective problems) to investigate the characteristics of DOPs and to provide a formal definition for DOPs. However, in other chapters, I will focus only on the empirical/experimental aspects of EDO and on solving only some specific subsets of DOPs. Specifically, in Chapters 5 and 6 I am interested in solving single-objective continuous dynamic constrained problems (DCOPs) and in Chapter 7 I am interested in solving single-objective continuous dynamic time-linkage problems (DTPs) (descriptions of these classes of problems will be provided in details in the corresponding chapters).

1.3 General research questions

The approach of this thesis is to start from some very general questions to get an overview of the important gaps in the field. Once the gaps have been identified, more specific questions will be raised and the thesis will focus on finding the answers to these specific questions.

The general questions that I am interested in finding the answer are:

> *What are the links between academic EDO research and real-world scenarios? Is there any type of problem, or any types of problem characteristics that are common in practical situations but have not been studied in EDO academic research?*

Answering the questions above requires us to carry out comprehensive literature review of the methods, performance measures, benchmark problems, and definitions on not only existing academic research in EDO, but also on real-world problems. These tasks will be dedicated to the next two chapters. Such a detailed literature review will provide us with knowledge about current gaps in EDO academic research to ask more specific questions such as:

> *If we have indeed found some classes of problems and problem characteristics that have not been considered in academic EDO research, what are the most important ones that we should study and why?*

The answers to such questions will help me to choose some of the most important type of problem/characteristics to study further in the thesis. For each problem/problem characteristic, further questions are asked in Subsection 3.1:

> *How can we capture these problems and characteristics in academic benchmark problems? What would be the performance of existing methods on these problems? How can we evaluate the performance of existing methods? What can we do to improve the performance?*
>
> *More importantly, how can we* ***effectively solve*** *these problems, which have not been solved by EDO before?*

These specific questions show us the research directions to be done in the rest of the thesis.

1.4 Outline of the thesis

This thesis is an attempt to answer the questions above. It is organised as follow:

Chapter 2 reviews and categorises existing EDO research about the solving methods, performance measures, and benchmark problems from the literature. The purpose of the chapter is to discuss the strengths and weaknesses of each method and more importantly to identify the current assumptions of the community about the characteristics of DOPs.

Chapter 3 follows by reviewing a large, representative set of recent real-world DOPs. The purpose of this chapter is to investigate for the first time some insights about the link between academic EDO research and certain classes of real-world DOPs and from that identify any gap between EDO academic research and real-world problems. Based on the review, in this chapter I will discuss the necessity and possibility to extend current EDO research to better reflect the common characteristics of DOPs and to solve wider ranges of DOPs more effectively. The chapter also sets out the research topics (for the rest of the thesis) to close some of the gaps that it has found: problem definition, continuous dynamic constrained optimisation, and dynamic time-linkage optimisation.

One of the gaps identified in Chapter 3: the lack of a formal definition to fully represent DOPs, is addressed in Chapter 4. In this chapter a new definition framework is proposed to (i) distinguish DOPs from other types of time-dependent problems; (ii) encapsulate the behaviours and types of dynamics; (iii) encapsulate the changing factors; and (iv) separate the static factors

from the dynamic factors.

Chapter 5 investigates one of the important but not yet well-studied classes of DOPs: dynamic constrained optimisation problems (DCOPs). In this chapter I will firstly present my investigations on the characteristics that might make dynamic constrained problems difficult to solve by some of the existing dynamic optimisation (DO) and constraint handling (CH) algorithms. I will then introduce a set of numerical dynamic benchmark problems with these characteristics and a set of performance measures to evaluate the performance of algorithms in DOPs/DCOPs. To verify my hypothesis about the difficulty of DCOPs, I will test several representative DO and CH strategies on the proposed benchmark problems. Based on the experiments I will also study some interesting observations where the presence or combination of different types of dynamics and constraints might make the problems easier to solve for certain types of algorithms. Based on the analysis of the results, I will propose a list of possible requirements that an algorithm should meet to solve DCOPs effectively.

Based on the results from Chapter 5, in Chapter 6 I will propose a set of new mechanisms to effectively handle dynamics in DCOPs and use them to develop new algorithms for solving DCOPs. The goal is to combine the advantages of DO and CH strategies while overcoming the drawbacks of these methods in solving DCOPs. To evaluate the performance of the new algorithms, I will compare them against several representative DO and CH algorithms using the set of benchmark problems proposed in Chapter 5. In this chapter I will also (i) carry out detailed analyses of how and why the newly proposed mechanisms/algorithms work better in DCOPs, (ii) investigate the contribution of each of the proposed mechanisms and (iii) study the influence of different parameter values on algorithm performance in solving DCOPs. These analyses reveal some interesting and counter-intuitive findings about the characteristics of DCOPs and the way we can solve DCOPs.

Chapter 7 focuses on another important but not yet well-studied classes of DOPs: dynamic time-linkage problems (DTPs). In this chapter I will identify a new and challenging class of DTPs where it might not be possible to solve the problems using the traditional methods. An approach to solve this class of problem under certain circumstances will be suggested and experiments to verify the hypothesis will be carried out. Two time-linkage benchmark problems will also be proposed to simulate the property of this new class of DTPs.

Chapter 8 concludes the thesis. Contributions of the thesis are summarised and future

research directions are also suggested.

1.5 Publications resulting from this thesis

Refereed or submitted journal papers

1. T. T. Nguyen and X. Yao (2010). Continuous Dynamic Constrained Optimisation - The Challenges, submitted to *IEEE Transactions on Evolutionary Computation*. (**given the option to revise for acceptance**).
2. T. T. Nguyen and X. Yao (2010). Solving Dynamic Constrained Optimisation Problems Using Repair Methods, submitted to *IEEE Transactions on Evolutionary Computation*. (**given the option to revise for acceptance**).

In-preparation journal papers

3. T. T. Nguyen, J. Branke, T. Ray and X. Yao (2010). Characteristics of dynamic optimisation problems: from academic evolutionary research to real-world problems. To be submitted to *IEEE Transactions on Evolutionary Computation* in October.
4. T. T. Nguyen, J. Branke and S. Yang (2010). Evolutionary Optimisation in Dynamic and Uncertain Environments: A Survey. (invited paper). To be submitted to *Swarm and Evolutionary Computation* in October

Refereed conference papers

5. T. T. Nguyen and X. Yao (2009). Benchmarking and Solving Dynamic Constrained Problems, Proceedings of the IEEE Congress on Evolutionary Computation CEC2009, Trondheim, Norway, 18-21 May 2009, IEEE Press, pp.690-697.
6. T. T. Nguyen and X. Yao (2009). Dynamic Time-linkage Problems Revisited. In M. Giacobini et al (Eds.), Proceedings of the 2009 European Workshops on Applications of Evolutionary Computation, EvoWorkshops 2009, Lecture Notes in Computer Science, Vol. 5484, Springer, pp.735-744.
7. H. K. Singh, A. Isaacs, T. T. Nguyen, T. Ray and X. Yao (2009). Performance of Infeasibility Driven Evolutionary Algorithm (IDEA) on Constrained Dynamic Single Objective Optimization Problems, Proceedings of the IEEE Congress on Evolutionary Computation CEC2009, Trondheim, Norway, 18-21 May 2009, IEEE Press, pp.3127-3134.

Technical report for the CEC'2009 competition on dynamic optimisation

8. Li C., Yang S., Nguyen T.T., Yu E.L., Yao X., Jin Y., Beyer H.-G. and Suganthan P.N. (2008). Benchmark Generator for CEC 2009 Competition on Dynamic Optimization, Technical report, University of Leicester and University of Birmingham, UK.

The following lists materials (or part) of the publications presented in the thesis:

- Chapter 2: publications [3, 4]
- Chapter 3: publication [3]
- Chapter 4: publications [6, 8]
- Chapter 5: publications [1, 5, 7]
- Chapter 6: publication [2]
- Chapter 7: publication [6]

Chapter 2

Literature review on evolutionary dynamic optimisation research

In this chapter I will focus on the empirical/experimental aspects of EDO research, covering some representative approaches (especially those in the continuous domain) in developing algorithms, generating benchmark problems and measuring algorithm performance. The purpose of the chapter is to discuss the strengths and weaknesses of each method and more importantly to identify the current assumptions of the community about the characteristics of DOPs.

Because the topics in EDO are very broad and diverse, it is impossible to cover everything in a chapter but only the topics that are most relevant to my research questions. The topics that will not be fully covered in this chapter are the classifications of DOPs and theoretical works. For a detailed literature review on classification methods for DOPs, readers are referred to my technical report in (Nguyen 2007). For details of theoretical works in the field, readers are referred to the works in (Wolpert & Macready 1997, Stanhope & Daida 1999, Rowe 1999, Ronnewinkel *et al.* 2000, Rowe 2001, Droste 2002, Droste 2003, Liekens *et al.* 2003, Liekens 2005, Rowe 2005, Arnold & Beyer 2006, Rohlfshagen *et al.* 2009, Tinos & Yang 2010).

2.1 Optimisation approaches

2.1.1 The goals of dynamic evolutionary algorithms

In stationary optimisation, in most cases the only goal of optimisation algorithms is to find the global optimum as fast as possible. However, in current EDO research where the considered problems are time-varying, the goal of an algorithm turns from finding the global optimum to firstly detecting the changes and secondly tracking the changing optima (local optima or ideally the global optimum) over time. In addition, in case the problem-after-change somehow correlates with the problem-before-change, an optimisation algorithm also needs to learn from its previous search experience as much as possible to hopefully advance the search more effectively. Otherwise, the optimisation process after each change will simply become the process of solving a different problem starting with the old population/structure.

The following sections will briefly review typical approaches in EDO that have been proposed to satisfy the goals above. We will discuss the strengths and weaknesses of the approaches and their suitability for different types of problems.

2.1.2 Introducing diversity when changes occur

Overview

In stationary optimisation, the convergence of an evolutionary algorithm is required so that the algorithm can focus on finding the best solution in the promising area that it has already found. In dynamic optimisation, however, convergence may result in negative effects. This is because if the dynamic landscape changes in one area and there is no member of the algorithm in this area, the change will become undetected. As a result, it is impossible for a normal EA to detect a change once it has already converged.

Intuitively one simple solution to this drawback is to increase the diversity of an EA after a change has been detected. This solution is described in the pseudocode of Algorithm 1.

Pioneer studies following this solution are Hyper-mutation (Cobb 1990) and Variable Local Search (VLS) (Vavak *et al.* 1997*b*, Vavak *et al.* 1998). They are different mostly in the step 2c (Algorithm 1) where different strategies were used to introduce diversity to the population. In his research, Cobb (1990) proposed an adaptive mutation operator called hyper-mutation whose mutation rate is a multiplication of the normal mutation rate and a hyper-mutation factor. The

Algorithm 1 Introducing diversity after detecting a change

1. *Initialise:*: Initialise the population
2. *For each generation*
   (a) *Evaluate:*:Evaluate each member of the population
   (b) *Check for changes*: Detect changes in the landscape by monitoring possible signs of changes, for example a reduction in the fitness of the best individuals
   (c) *Increase diversity*: If there is any change, increase the diversity of the population by adaptively change the mutations (sizes or rates) or relocate individuals
   (d) *Reproduce*: Reproduce a new population using the adjusted mutation/learning/adaptation rate
   (e) Return to step 2a

hyper-mutation is invoked only after a change is detected.

In the Hyper-mutation method, the fact that the step size of the mutation size is not adaptive may decrease the performance of the EA. To improve this, in Variable Local Search Vavak *et al.* (1996) provided a mechanism to control the size of mutation by defining a variable local search range. This range is determined by the formula $(2^{BITS} - 1)$ where BITS is a value adjustable during the search.

In their consecutive paper Vavak *et al.* (1997*b*) improved VLS by making it adaptive using a learning strategy borrowed from the feature partitioning algorithm Guvenir and Sirin (1993). For each individual, the learning strategy learns to map the severity of the change with a suitable local search range selected from a pre-defined set, then classify individuals into disjoint sets according to their local search range (see Figure 2.1).

An interesting way of introducing diversity was proposed in (Yu & Suganthan 2009) where the individuals to be introduced to increase diversity are not randomised ones but some previous good solutions (which have been specifically chosen so that they either are most diversified (to be used when the algorithm prematurely converged) or represent different parts of the search space instead of getting crowded in one area (to be used when a change is detected)). In other words, in this work the diversity-introducing and memory approaches are combined into one to increase diversity while still be able to recall previous good solutions.

Recently diversity-introducing has been used to handle dynamic constraints. In (Nguyen & Yao 2010*b*), hyper-mutation was used with a change detection method in an EA to solve

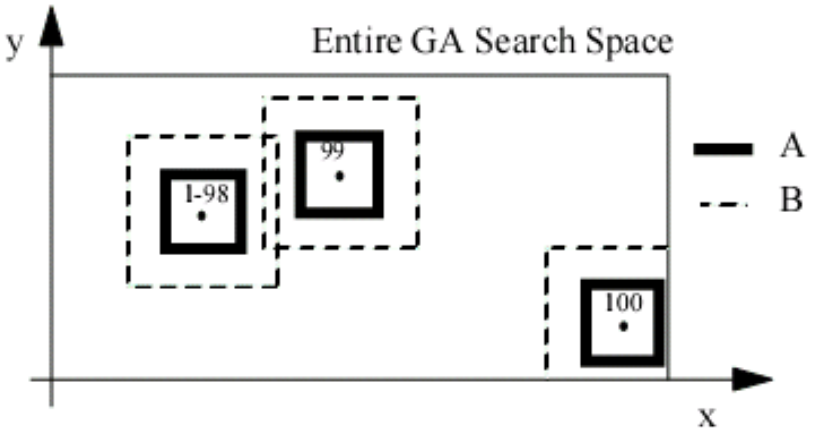

Figure 2.1: This figure (reproduced from (Vavak *et al.* 1998)) shows an example of applying VLS operator to a population of 100 individuals. Assume that before the change the first 98 individuals of the population converged to the global optimum. After invoking the VLS operator, the search range of individuals are restricted to only the hypercubes marked A. If the search is not successful the search range may be extended to the hypercubes B. Note that the search range of each group of individuals might be different from each other

dynamic constraint problems. Detectors are placed near the boundary of feasible regions and when the feasibility of these detectors change, the EA increases its mutation rate to raise the diversity level to track the moving feasible regions. The mutation rate is decreased once the moving feasible regions is tracked successfully.

Diversity-introducing approach is also used in dynamic multi-objective optimisation (DMO). For example, in a multi-population algorithm for DMO (Goh & Tan 2009*a*), when a change is detected stochastic individuals and some competitor individuals from other sub-populations are introduced to each sub-population to increase diversity.

The approach of introducing diversity after changes is also used in Particle Swarm Optimisation (PSO). Hu & Eberhart (2002) introduced a simple mechanism in which a part of the swarm or the whole swarm will be re-diversified using randomization after a change is detected. Janson & Middendorf (2005) and Janson & Middendorf (2006) followed a more sophisticated mechanism where after each change the swarm is divided into a hierarchy of several sub-swarms for a certain number of generations. The purpose of this hierarchy is to prevent the swarm from converging to the old position of the global optimum, which might have been moved since the last change.

Recently Woldesenbet & Yen (2009) proposed a new adaptive method named "relocation

variable", which can be considered belonging to the class of introducing diversity/adaptability approaches. In this method, after a change individuals are relocated based on the changes in their function values and on the sensitivities of their coordinations to changes. Specifically, based on the history of their performance a relocation radius is estimated for each individual. The individuals will be relocated (mutated) to a position within this radius for a number of times and the best fit position will be used as a member for the new population. The size of each relocation radius depends on the sensitivity of the individual to changes in the environment. The more sensitive the individual is, the larger the radius.

The introducing-diversity approach is still commonly used in many recent EDO algorithms, for example (Parrott & Li 2006, Moser & Hendtlass 2007*a*, Richter 2009, Richter & Yang 2009, Richter 2010, Nguyen & Yao 2010*b*).

Strengths and weaknesses

In general methods following this approach appear to be good in solving problems with continuous changes where changes are small and medium. This is because invoking mutations or distributing individuals around an optimum resembles a type of "local search", which is useful to observe the nearby places of this optimum. Thus if the optimum continuously moves to nearby places, it might be tracked (Vavak *et al.* 1996, Vavak *et al.* 1997*b*).

However, this approach has some drawbacks that might make it not so suitable for certain type of problem. They are listed bellows:

- *Dependence on whether changes are known / easy to detect or not*: Because most methods following this approach detect changes by observing the reduction of fitness of some best performers and/or the population as an indication of changes, if a change appears in a place where no individual exist, it will go undetected (Morrison 2004). For example, it has been shown that in dynamic constrained problems, the diversity-introducing strategy cannot detect changes when the dynamic constraints expose new global optima without changing the fitness value of the previous optima (Nguyen & Yao 2009*a*, Nguyen & Yao 2010*a*)

- *Difficulty in identifying the correct mutation size (in case of Hyper-mutation and VLS) or the number of sub-swarms (in case of Hierarchy PSO)*: too small steps will resemble local search while too large steps will result in random search (Jin & Branke 2005).

- *Not being effective for solving problems with random changes or large changes (changes are severe)*: Because many diversity-introducing methods have their mutation/ relocation size restricted to a specific range, it is obvious that they are not effective if either the changes are larger than this range or the changes are random. For example, experiments show that Hierarchy PSO may not perform as good as traditional PSO in tested problems with large changes (Janson & Middendorf 2006).

- *Not being effective for solving problems with fast changes*: After introducing diversity, methods following this approach need time to converge again. As a result, if the change is fast, they may not be able to find the global optimum (Cobb 1990).

2.1.3 Maintaining diversity during the search

Overview

Another approach in solving dynamic problems is to maintain population diversity throughout the search process to avoid the possibility that the whole population converge into one place, hence unable to either track the moving optimum or detect a new competing peak (see Algorithm 2).

Algorithm 2 Pseudo code for algorithms that maintain diversity

1. *Initialise:*: Initialise the population
2. *For each generation*
   (a) *Evaluate:*:Evaluate each member of the population
   (b) *Maintain diversity*: Add a number of new, diversified individuals to the current population to increase diversity
   (c) *Reproduce*: Reproduce a new population
   (d) Return to step 2a

Methods following this approach do not detect changes explicitly. Instead they rely on their diversity to adaptively cope with the changes. Typical examples of this approach are Random Immigrants (Grefenstette 1992), fitness sharing (Andersen 1991), Thermo-Dynamical GA(Mori *et al.* 1996), Sentinel Placement (Morrison 2004), Population-Based Incremental Learning (Yang & Yao 2005), several Particle Swarm Optimisation variants (Janson & Middendorf 2005, Blackwell & Bentley 2002, Blackwell & Branke 2006, Blackwell 2007) and dynamic Evolutionary

Multiobjective optimisation (Bui *et al.* 2005, Abbass & Deb 2003, Toffolo & Benini 2003).

In the Random Immigrants method, in every generation a number of generated random individuals are added to the population to maintain diversity. Experimental results show that the method is more effective in handling dynamics than the regular EA (Grefenstette 1992). It is reported that the high diversity level brought by random immigrants also helps in handling constraints. In (Nguyen & Yao 2010*b*) it was shown that when combined with the constraint-handling repair method, random-immigrant significantly improve the performance of the tested EA.

Morrison (2004) follows a slightly different mechanism in which instead of generating random individuals, his Sentinel Placement method initialises a number of sentinels which are specifically distributed throughout the search space. These sentinels can still participate in the reproduce process of the population (to maintain diversity) but will never be removed (so that they can always track possible coming changes). Experiments show that this method might get better results than Random Immigrants and Hyper-mutation in problems with large and chaotic changes (Morrison 2004).

Two other approaches - Parallel PBIL (PPBIL2) and Dual PBIL (DPBIL) were proposed by Yang & Yao (2005). These methods are based on the Population-based Incremental Learning (PBIL) algorithm, which is a simple combination of population-based EA and incremental learning. PBIL has an adjustable probability vector which is used to generate individuals. After each generation the probability vector is updated based on the best found solutions. It ensures that the vector will gradually "learn" the appropriate value to generate high quality individuals. A pseudo code of PBIL was shown in Algorithm 3.

In PPBIL2, Yang & Yao (2005) improved PBIL for dynamic optimisation by maintaining two parallel probability vectors. The first one is a central initialised probability vector similar to that of normal PBIL. The second one is a random initialised probability dedicated to maintaining diversity during the search. The two vectors are sampled and updated independently. On initial they have the same sample size. However, throughout the search their sample sizes might be adjusted based on their relative performance.

Although PPBIL2 might offer better diversity than its original version PBIL, in certain cases where the intervals between changes are large, the two populations may still end up in convergence and the algorithm still lose diversity. As a result, Yang & Yao (2005) proposed another

Algorithm 3 Population-based Incremental Learning

1. *Initialise:*: Initialise P, the probability vector: $P[i] = 0.5$, $i = 1, ...n$ where n is the number of variables
2. *For each generation:*
   (a) *Generate*: Generate each individual by sampling the space using the probability vector P
   (b) *Evaluate*: Evaluate each member, assign B to the best individual found
   (c) *Update P*: Update the vector P based on the best individual B: $P[i] = (1 - \alpha) P[i] + \alpha B[i]$ where α is the pre-defined learning rate.
   (d) Return to step 2a

improved version of PBIL, the DPBIL. Similar to PPBIL2, DPBIL also has two probability vectors. However these vectors are *dual* with each other, which means that given the first vector P_1, the second vector P_2 is determined by $P_2[i] = 1 - P_1[i]$, $i = 1, ..., n$ where n is the number of variables. During the search only P_1 needs to learn from the best generated solution because P2 will change with P_1 automatically. PBIL and dual PBIL were also combined with random-immigrants in (Yang & Yao 2008) with better results than the original algorithms.

The approach of maintaining diversity is also used in Particle Swarm Optimisation (PSO) to solve dynamic continuous problems. In their charged PSOs (Blackwell & Bentley 2002, Blackwell & Branke 2006, Blackwell 2007), Blackwell *et al.* applied a *repulsion* mechanism, which is inspired by the atom field, to prevent particles/swarms to get too closed to each other. In this mechanism, each swarm is comprised of a nucleus and a cloud of charged particles which are responsible to maintain diversity. There is a repulsion among these particles to keep particles from approaching near to each other.

Bui *et al.* (2005) proposed another interesting way to maintain diversity in dynamic optimisation: using multi-objective approaches. The dynamic problem is represented as a two-objective function. The first one is the original single objective, and the second is a special objective created to maintain diversity. Other examples of using multiple objectives to maintain diversity can be found in Abbass & Deb (2003) and Toffolo & Benini (2003), where six different following types of objectives were proposed:

- Retain more old solutions (favour old individuals based on an attached time stamp)
- Retain more random solutions

- Slow down the convergence by reversing the optimisation of the first objective
- Keep a distance from the closest neighbor
- Keep a distance from all individuals
- Keep a distance from the best individual of the population

The diversity-maintaining strategy is still the main strategy in many recent approaches, for example (Janson & Middendorf 2005, Yang & Yao 2005, Blackwell & Branke 2006, Blackwell 2007, Yang & Yao 2008, Deb *et al.* 2007, Riekert *et al.* 2009, de França & Von Zuben 2009, Cheng & Yang 2010).

Strengths and weaknesses

Methods following this approach can bring the following advantages:

- *May be good for solving problems with severe changes*: Thanks to its good diversity, in certain situations the approach is good to solve problem with large changes (for example in (Nguyen & Yao 2009*a*, Nguyen & Yao 2010*b*, Nguyen & Yao 2010*a*) it has been shown that random-immigrants help significantly improve the performance in dynamic constrained problems where changes are severe due to the presence of disconnected feasible regions)
- *May be good for solving problem with slow changes* (as shown in e.g. (Andersen 1991, Yang & Yao 2005)). This is because for slow changes an algorithm with high diversity may have enough time to converge.
- *May be effective in solving problems with competing peaks* (as reported in (Cedeno & Vemuri 1997))

However, methods that maintain diversity through out the search also have some disadvantages as follow:

- *Slow*: Continuously focusing on diversity may slows down, or even distract the optimisation process (Jin & Branke 2005).
- *Not effective when the changes are small*: Most methods following this approach maintain their diversity by adding some stochastic element through out the search space. Obviously

it will make the algorithm harder to track small changes where the optima just take a slight move away from their previous places (Cobb & Grefenstette 1993).

2.1.4 Memory Approaches

When changes in dynamic problems are periodical or recurrent, i.e. the optima may return to the regions near their previous locations, it might be useful to re-use previous found solutions to save computational time and to bias the search process. To re-use old solutions in this manner, many researchers decide to add some types of memory components to their EAs to make it more effective in tracking periodical changes. The memory can also play the role as a reserved place storing old solutions for maintaining diversity when needed. The memory can be integrated *implicitly* as a redundant representation in the EAs, or it could be maintained *explicitly* as a separate memory component.

Implicit memory

Redundant coding using diploid genomes are the most common implicit memory used in EAs for solving dynamic problems e.g. (Goldberg & Smith 1987, Ng & Wong 1995, Lewis *et al.* 1998, Uyar & Harmanci 2005, Yang 2006*c*). A diploid EA is usually an algorithm whose chromosomes contain two alleles at each locus. Although most normal EAs for stationary are haploid, it is believed that diploid, and other multiploid approaches, are suitable for solving non-stationary problems (Lewis *et al.* 1998). A pseudo code for multiploid approaches for dynamic environments is described in Algorithm 4.

Algorithm 4 Multiploid EA for dynamic optimisation

1. *Initialise:*: Initialise the population and the multiploid representation
2. *For each generation*
   (a) *Evaluate:* Evaluate each member of the population
   (b) For each individual:
      i. *Check for changes*: detect any change in the fitness that may be caused by a change in the landscape
      ii. *Adjust the dominance level of each allele* : If there is any change, adjust the dominance to accommodate the current change
      iii. *Select the dominant alleles according to their dominance level*
   (c) *Reproduce*: Reproduce a new population using the adjusted mutations
   (d) Return to step 2a

As can be seen in Algorithm 4, in order to design a multiploid EA, we need to take into account three tasks:

1. represent the redundant code;
2. represent/adjust the dominance of alleles; and
3. check for changes in the landscape.

One typical way to represent the dominance of alleles is to use a table (Ng & Wong 1995, Ryan 1996) or a mask (Collingwood *et al.* 1996) mapping between genotypes and phenotypes. The dominance then can be changed adaptively among alleles depending on the detection of changes in the landscape. To detect changes, we can use several methods, for example checking the change in the fitness of an individual (Ng & Wong 1995, Lewis *et al.* 1998); or using an incremental learning probability (Yang 2006*c*).

Some other examples of algorithms following the approach of using the environment as implicit memory are the studies of (Guntsch & Middendorf 2002), (Guntsch *et al.* 2000) and (Randall, 2005).

Explicit memory

Methods that maintain the memory explicitly are described by the pseudo code in Algorithm 5:

Algorithm 5 EA for dynamic optimisation with explicit memory

1. *Initialise:*:
   (a) Initialise the population
   (b) Initialise the explicit memory
2. *For each generation*
   (a) Evaluate each member of the population
   (b) Update the memory
   (c) Reproduce a new population
   (d) Use information from the memory to update the new population
   (e) Return to step 2a

Methods following the approach of using explicit memory need to accomplish four tasks:

1. *Decide the content of the explicit memory*: The content of the memory can be either:

(a) *Direct memory:* In most cases the direct memories are the previous good solutions (Louis & Xu 1996, Mori *et al.* 1998, Branke 1999, Bendtsen & Krink 2002, Yang 2005*a*, Yang 2006*a*, Zeng *et al.* 2007, Yang & Yao 2008, Yu & Suganthan 2009). In (Yu & Suganthan 2009) for certain circumstances the most diversified solutions (in term of standard deviation of fitness) are also selected for the memory.

(b) *Associative memory:* Various type of information can be included in the associative memory, for example the information about the environment at the considered time (Ramsey & Grefenstette 1993), (Eggermont *et al.* 2001); the list of environmental states and state transition probabilities (Simões & Costa 2008); the probability vector that created the best solutions (Yang & Yao 2008); the distribution statistics information of the population at the considered time (Yang 2006*a*); the probability of the occurrence of good solutions in each area of the landscape (Richter & Yang 2008) (Richter & Yang 2009); or the probability of likely feasible regions (Richter 2010)

2. *Decide how to update the memory:* Generally the best found elements (direct or associative) of the current generation will be updated to the memory. These newly found elements will replace some existing elements in the memory, which can be one or some of the followings:

(a) The oldest member in the memory (Trojanowski & Michalewicz 1999, Eggermont *et al.* 2001, Simões & Costa 2007, Woldesenbet & Yen 2009)

(b) The one with the least contributions to the diversity of the population (Branke 1999, Eggermont *et al.* 2001, Yang 2005*a*, Simões & Costa 2007, Yang & Yao 2008). One common way to evaluate this criterion is to examine the similarity of elements in the memory, for example evaluating the minimum distance among all pairs of memory elements (Branke 1999, Simões & Costa 2007). In this case the less fit one of a pair will be replaced.

(c) The one with least contribution to fitness (Eggermont *et al.* 2001)

3. *Decide when to update the memory*: Ideally if we know exactly when a change happens, then the most suitable time to update the memory is right after the time the change happens. However in general it might not always be possible to know exactly when a

change happens. As a result the memory may also be updated after each generation or after a certain number of generations. Doing so might also favour diversity, for example see (Branke 2001*a*, Branke 2003, Yu & Suganthan 2009).

4. *Decide how to use the memory*: Usually the best elements in the memory (i.e. the ones that show the best results when being re-evaluated) will be used to replace the worst individuals in the population. Replacement can take place after each generation or after a certain number of generations, or it can be done after each change if the change can be recognised.

Memory is used not only in the EA and swarm-based methods as mentioned above, but also in Artificial Immune Systems (AIS) (Simões & Costa 2003), (Yang 2006*b*). This approach is inspired by the biological immune systems, which are able to identify the correct types of harmful antigens, hence produce the appropriate antibodies to destroy the antigens. In AIS approaches, changes in the dynamic environments are usually viewed as antigens and the "building blocks" (gene segments) from successful individuals in the past are considered as antibodies. The gene segments are stored in a gene library so that they can be recalled whenever a change occurs. To identify which gene segments (antibodies) should match with a particular antigen (change in the environment), each individual in the gene library is associated with the average fitness of the population at the moment it was stored. This value is used as an identification tag to know which element from the library should be used when a change is discovered. The one whose attached fitness value is most similar to the current averaged fitness will be chosen (Simões & Costa 2003). The previous memory-based AIS studies have been made only on the 0/1 dynamic knapsack problem (Simões & Costa 2003) and the binary encoded test problems (Yang 2006*b*).

Strengths and weaknesses

Here are the advantages of using memory-based approaches:

1. *Effective for solving problems with periodically changing environments.* Thanks to their ability to recall old solutions from the memory, memory-based approaches are especially suitable for solving problems with periodical changes. For example, Yang (2008) showed that the memory-based versions of GA and random-immigrant significantly outperform the original algorithms in cyclic dynamic environments.

2. *May be good in slowing down convergence and favour diversity* (Branke 2001*a*), (Branke 2003).

Memory approaches, however, also have some disadvantages that may require them to be integrated with some other methods for the best results:

1. *Might be useful only when optima reappear at their previous locations or if the environment returns to its previous states.* This might be the most significant disadvantage of memory-based approaches. In his experiments Lewis *et al.* (1998) showed that redundant coding does not ensure enough diversity to adaptively for random changes or oscillated changes with one or more target has been changed during the search. Branke (1999) also generalized the same weakness in some explicit memory approaches, pointing out that the memory might no longer be effective if the oscillation does not bring the global optimum to the exact previous location but a slightly different one (Branke 2001*a*). He then concluded that memory alone could not be enough for dynamic optimisation. It should be combined with some type of diversity methods.

2. *Might not be good enough to maintain diversity for the population*, as pointed out by (Branke 1999). Recently some studies have tried to improve this disadvantage by combining memory-based approaches with diversity schemes e.g. (Simões & Costa 2007) (Yang 2008).

3. *Redundant coding approaches might not be good for cases where the number of oscillating states is large.* There are two reasons for this:

   (a) Firstly, the redundant code might become too large, hence reduce the performance of the algorithm. In order to recover information about a previous state of the environment, redundant coding approaches need to encode the information about this state into the representation. The larger the number of changing states that a problem has, the larger the number of codes needed for representing the changing states. For example, if an environment oscillates between two states, we need a diploidy solution. If there are three states, then we may need a triploid solution. Experiments (Lewis *et al.* 1998) also shows that a diploid approach may be able to

adapt and switch between only two states. If there are more than two states, the approach may fail (Branke 2001*a*).

(b) Secondly, in practice it might not always be possible to know the number of oscillating states before hand. Without this information, it is impossible to design an appropriate representation for the redundant code.

4. *The information stored in the memory might become redundant (and obsolete) when the environment changes.* This redundancy may affect the performance of the algorithm. For example, Branke (2001*b*) empirically showed that memories are of no use if there is no recurrence in the environments.

2.1.5 Prediction Approaches

In certain cases, changes in dynamic environments may exhibit some patterns that are predictable. In this case, it might be sensible to try to learn these types of patterns from the previous search experience and based on these patterns try to predict changes in the future. Some studies has been made following this idea to exploit the predictability of dynamic environments. Obviously, memory approaches, which are proposed to deal with periodical changes, can also be considered a special type of prediction approaches. However, generally methods following the prediction approach are able to use their memory to cope with more various types of changes than only cyclic/recurrent changes. A pseudo code describing prediction approaches is shown in Algorithm 6.

One of the first research on predicting changes might be the study of Ramsey & Grefenstette (1993). The authors proposed a method to represent the characteristics of the environment in several variables so that a learning method (case-based reasoning) can be used to map between these variables and the found optima. The mapping information is then used by the algorithm to identify the type of the landscape after a change and introduce the suitable old solutions accordingly to the population. A similar learning/classification approach was also carried out by Eggermont *et al.* (2001).

A common prediction approach is to predict the movement of the moving optima. Hatzakis & Wallace (2006) combined a forecasting technique (Autoregressive) with an EA. This forecasting technique is used to predict the location of the next optimal solution after a change is detected. The forecasting model (time series model) is created using a sequence of optimum positions

Algorithm 6 EA following the prediction approach to solve dynamic problems

1. *Initialise phase*:
   (a) Initialise the population
   (b) Initialise the learning model and training set
2. *Search for optimum solutions and detect changes*
3. *If a change is detected*
   (a) Use the current environment state as the input for the learning model
   (b) Use the learning model to estimate the type of this current change and/or how the next change should be
   (c) Generate new individuals/recall old individuals that best matches with the estimation
   (d) Search for the new optimum using the new population
   (e) Update the training set based on the search results
4. Return to step 2

found in the past. Experimental results show that if this algorithm can predict the movements of optima correctly, it can work well with very fast changes. A similar research was proposed in (Rossi *et al.* 2008) where the movement of optima was predicted using Kalman filters. The predicted information (the next location of the optimum) is incorporated into an EA in three ways: First, the mutation operator is modified by introducing some bias so that individuals' exploration is directed toward the predicted region. Second, the fitness function is modified so that individuals close to the estimated future position are rewarded. Third, some "gift" individuals, which are generated at the predicted positions, are introduced to the population to guide the search. Experiments on a visual tracking benchmark problem show that the proposed method does improve the tracking of the optimum, both in terms of distance to the real position and smoothness of the tracking.

Another approach is to predict the locations that individuals should be re-initialised when a change occurs. In (Zhou *et al.* 2007) this approach is used to solve two dynamic multi-objective optimisation benchmark problems in two ways: First, the solutions in the Pareto set from the previous change periods were used as a time series to predict the next re-initialisation locations. Second, to improve the chance of the initial population to cover the new Pareto set, the predicted re-initialisation population is perturbed with a Gaussian noise whose variance is estimated based

on history data. Compared with random-initialisation, the approach was able to achieve better results on the two tested problems. Another approach to estimate the areas to re-initialise individuals after a change occurs is the relocation variable method (Woldesenbet & Yen 2009) described in Subsection 2.1.2. This method to some extent can also be considered a prediction method.

Another interesting approach is to predict the future moment when the next change will occur and which possible environments will appear in the next change (Simões & Costa 2008, Simões & Costa 2009). In these work, the authors used two prediction modules to predict two different factors. The first module, which uses either linear regression (Simões & Costa 2008) or non-linear regression(Simões & Costa 2009), is used to estimate the generation when the next change will occur. The second module, which uses a Markov chain, monitors the transitions of previous environments and based on this data provides estimations of which environment will appear in the next change. Experimental results show that an EA with the proposed predictor is able to perform better than a regular EA in cyclic/periodic environments.

Relating to prediction approaches, recently there are also some studies (Bosman 2005, Bosman 2007, Bosman & Poutré 2007, Nguyen & Yao 2009*b*) on time-linkage problems, i.e. problems where the current solutions made by the algorithms can influence the future dynamics. In such problems, it was suggested that the only way to solve the problems effectively is to predict future changes and take into account the possible future outcomes when solving the problems online. Another related study is the anticipation approach (Branke & Mattfeld 2005) in solving dynamic scheduling problems where in addition to finding good solutions, the solver also tries to move the system "into a flexible state" where adaptation to changes can be done more easily. Specifically, because it is observed that in the tested dynamic job-shop scheduling problem, the flexibility of the system can be increased by avoiding early machine idle times, the authors proposed a scheduling approach where in addition to the main optimality objective, solutions with early idle time are penalised. The experimental results show that such an anticipation approach significantly improved the performance of the system.

Strengths and weaknesses

Methods following the prediction approach may become very effective if their predictions are correct. In this case, the algorithms can detect/track/find the global optima quickly, as shown

in (Hatzakis & Wallace 2006), (Yang 2006*b*) and (Simões & Costa 2003).

However, prediction-based algorithms also have their own disadvantages, mostly due to training errors. These errors might be resulted from:

1. *Wrong training data*: If the algorithm has not performed successfully in the previous change periods, the history data collected by the algorithm might not be helpful for the prediction or might even provide the wrong training data.

2. *Lack of training data*: As in the case of any learning/predicting/forecasting model, the algorithms may need a large enough amount of training data to produce the best results. It also means that the prediction can only be started after a certain amount of time when the training data has been collected. For example, in the prediction-based methods e.g. (Simões & Costa 2008, Simões & Costa 2009, Bosman 2005, Bosman 2007), the prediction should only be done once the algorithms get enough training data. In the case of dynamic optimisation where there is a need of finding/tracking the optima as quick as possible, this might be a disadvantage.

3. *The nature of the dynamic problems*:

   - If changes in the dynamic environment are easily predictable (e.g. linear, periodical or deterministic), the result is expected to be good, as can be seen in (Hatzakis & Wallace 2006, Rossi *et al.* 2008).
   - However, if the changes are stochastic, then prediction approaches might not get satisfiable results. For example, Nguyen & Yao (2009*b*) illustrated a situation where historical data are actually inappropriate for the prediction and might even mislead the predictor to get worse results.

2.1.6 Making use of the self-adaptive mechanism of EAs

Another approach is to make use of the self-adaptive mechanisms of EAs to cope with changes. To some extent this approach closely relates to the prediction approach, because deep down self-adaptation is the outcome of a process involving learning and evolving based on history data.

One example is the GA with Genetic Mutation Rate (Grefenstette 1999), which allows the algorithm to evolve its own mutation strategy parameters during the search based on the fitness

of the population. In this method, the mutation rate is encoded in genes and is influenced by the selection process. The algorithm was tested in both gradual and abrupt dynamic landscapes. The results show that the algorithm have better performance than normal GA. However, it is still not better than hyper-mutation (see section 2.1.2 and (Cobb 1990)) - a method that increases its mutation rate after each change.

A similar method was proposed by Ursem in his Multinational Genetic Algorithm (MGA) (Ursem 2000). Five different parameters: probability for mutation, probability for crossover, selection ratio, mutation variance and distance are encoded in the genomes of his MGA for self-adaptation. The self-adaptation mechanism works well in simple cases where the velocity of moving peaks is constant. However, in cases where the velocity is not constant, the self-adaptation seems to be not fast enough. These two results show the difficulty of applying self-adaptive parameter tuning to complex dynamic optimisation.

Methods that adaptively evolve their strategies by learning from the environments to handle dynamics, like the VLS (Vavak *et al.* 1998), PBIL (Yang & Yao 2005) and variable relocation (Woldesenbet & Yen 2009) variants mentioned in the previous subsections, can also be categorised into this group of self-adaptive approaches.

Some researchers also express their interests in using the self-adaptive mechanism of such EAs as Evolution Strategy (ES) or EP (Evolutionary Programming) in dynamic optimisation. However, it has not been clear of whether the original self-adaptation mechanism of ES/EP alone can be used effectively in dynamic optimisation. Empirical experiments show mixed results. Angeline (1997) examined self-adaptive EP (saEP) and showed that the strategy is not effective for all types of tested problems. Bäck (1998) showed that the log-normal self-adaptation in ES may perform better than saEP in the same problems, however experiments also pointed out that the sensitivity of ES in dynamic environments is worse than its sensitivity in stationary environment (Salomon & Eggenberger 1997) and that ES might be unreliable in rapidly changing environment (Weicker & Weicker 1999). Weicker (2003) also argued that it is possible that the Gaussian mutation in the standard ES self-adaptation might not be appropriate for dynamic optimisation.

There are some mathematical analyses on the performance of self-adaptive ES on dynamic environments. Arnold & Beyer (2002) pointed out that the cumulative mutation strength adaptation of ES can work well on a variant of the sphere model with random dynamics of the

target. The strategy can realise optimal mutation strengths for the model. However, in the sphere modal with linear dynamics, another research of Arnold & Beyer (2006) revealed that the mutation strength realised by ES is not the optimal one (but the adaptation still ensures that the target can be tracked).

Wang & Wineberg (2006) proposed a different approach, which can also be considered as a self-adaptive variant of EA. In this approach, the algorithm comprises of three types of populations: a population to search, a population to measure the efficiency of exploration (based on genotypic changes) and a population to measure the efficiency of exploitation (based on fitness improvements). By observing the two later populations, the algorithm will dynamically adjust the selection pressure to balance exploration and exploitation. Experiments in (Wang & Wineberg 2006) show that the new approach can track the global optimum better than GA with random immigrants in problems with optima moving linearly. There is however no report of whether the algorithm can work well in problems with periodical change or problems with random changes.

Another recent adaptability-introducing approach is the work of Yang & Richter (2009) where a mechanism to increase the learning rate of the Population-based Incremental Learning (PBIL) algorithm was proposed. As mentioned in Subsection 2.1.3, PBIL has an adjustable probability vector which is used to generate individuals. After each generation the probability vector is updated based on the best found solutions to make sure that the vector will gradually "learn" the appropriate value to generate good solutions. A high learning rate after a change as implemented in (Yang & Richter 2009) will help the algorithm to learn the suitable value for the vector faster and hence will be able to adapt to the new environment faster. After some generations the learning rate will be resumed to the normal value to make sure that the algorithm converges properly.

Recently Riekert *et al.* (2009) proposed an adaptive Genetic Programming to solve dynamic classification problems. The algorithm is made adaptive in several ways. First, the elitist proportion is adaptively increased when fitness is improved and vice versa. Second, the crossover rate is decreased to save computational effort when the performance is satisfactory. Otherwise the crossover rate is increased to generate more solutions. Third, when a change occurs the mutation rate is continually modified until it succeeded in finding good solutions.

2.1.7 Multi-population approaches

Overview

Another approach, which to some extent can be seen as a combination of diversity maintaining/introducing, memory and self-adaptation, is to implement multiple sub-populations concurrently. Each sub-population may handle a separate area of the search space. Each of them may also take responsibility for a separate task. For example, some sub-populations may focus on searching for the global optimum while some others may concentrate on tracking any possible changes. These two types of populations then may communicate with each other to bias the search. A pseudo code of multi-population approaches is listed out in Algorithm 7

Algorithm 7 Multi-population approach

1. *Initialise:*:
   (a) Initialise the setP_{search} of sub populations searching for the global optima
   (b) Initialise the set P_{track} of sub populations tracking changes in the landscape
2. *For each generation*:
   (a) *Search for optima:* the sub-populations in P_{search} search for the global optima
   (b) *Track changes*: the sub-populations in P_{track} track any changes
   (c) *Maintain diversity:*:Re-allocate/split/merge the sub-populations so that they are not overlapped and can cover a larger area of the search space
   (d) *Adjust*: Re-adjust each sub-population in P_{search} based on the experience from sub-populations in P_{track}
   (e) Reproduce each sub-population
   (f) Return to step 2a

As can be seen from the pseudo code in Algorithm 7, methods following the approach of using multiple populations usually need to accomplish two goals: First, they may need to assign different types of tasks to different sub-populations, for example P_{search} to search and P_{track} to track, so that the search can be done effectively. Second, they need to divide the sub-populations appropriately and make sure that the sub-populations are not overlapped to have the best diversity and also to avoid the situation where many sub-populations finding the same peak.

For the first goal, assigning different tasks to the sub-populations, different methods have

different approaches. One approach was proposed by Oppacher & Wineberg (1999) in their Shifting Balance Genetic Algorithm (SBGA). In SBGA, there are a number of small populations in P_{search} searching for new solutions and there is only one large population in P_{track} to track changing peaks.

Another method, the Self-Organizing Scouts (SOS) (Branke *et al.* 2000), follows a different direction where using the main large population to search for optima (P_{search}) and dedicating several small populations to track any change of each optimum that the algorithm has found so far (P_{track}). Whenever the main population finds a new peak, it creates a new sub-population to track changes in this peak. This approach was adopted in different types of EAs and meta-heuristics, for example GA (Cheng & Yang 2010), DE (Mendes & Mohais 2005, Lung & Dumitrescu 2007) and PSO (Blackwell 2007, Fernández & Arcos 2010). Relating to using one large population to search and a smaller to track changes, an algorithm named RepairGA for solving dynamic constrained problems was proposed in (Nguyen & Yao 2009*a*). In this method a large sub-population is dedicated to searching and one smaller sub-population is dedicated to tracking the moving feasible regions. The difference between RepairGA and previous approaches is that in RepairGA the two sub-populations are allowed to overlap in the search space because their main purpose is not to maintain diversity. What distinguishes the two sub-populations in this work is that the earlier accepts both infeasible and feasible solutions while the other contains only feasible solutions.

Another approach, the Multinational GA (MGA) introduced by Ursem (2000), integrates the functions of P_{search} and P_{track} into each sub-population. It means that each population can both search for new solutions and track changes. Whenever a sub-population detects a new optimum, it will split into two sub-populations to make sure that each sub-population only tracks one optimum at a time. This approach has been used not only in EAs but also in Artificial Immune algorithms, for example (de França & Von Zuben 2009). The approach is also used by PSO-based algorithms for dynamic optimisation. One example is the Speciation PSO (Li *et al.* 2006) whose each sub-population, or species, is a hyper-sphere defined by the best fit individual and a specific radius. Another recent PSO example that also have multi swarms with equal roles is the Clustering PSO in (Li & Yang 2009).

Also relating to the goal of assigning the tasks to sub-populations, it should be noted that in dynamic optimisation multiple populations are used not only for the purpose of exploring

different parts of the search space, but also for the purpose of co-evolution. In (Nguyen & Yao 2009*a*), a co-operative coevolution model was implemented where two sub-populations are maintained. One sub-population named reference population focuses on tracking the moving feasible regions while the other focuses on finding the global optimum. The two sub-populations co-evolve in a way that the former attracts the latter to promising regions while the latter informs the former about the appearance of any new feasible region. In (Goh & Tan 2009*a*), another co-evolution model was used. However, in this study the multiple populations were not used to explore different areas but to optimise different subcomponents, which are the decompositions of the solution vector.

For the second goal, *dividing the sub-populations and making sure that the sub-populations are not overlapped*, there are also different approaches. The most common approach is the clustering approach: choosing some solutions in the population as the centres of the future clusters, then defining each sub-population as a hyper-cube or sphere with a given size. All individuals within the range of a hyper-cube/sphere will belong to the corresponding sub-population of that hyper-cube/sphere. (Branke *et al.* 2000) is one the earliest methods that adopt this approach. SOS (Branke *et al.* 2000) keeps the sub-populations from overlapping by using an idea borrowed from the Forking Genetic Algorithm (FGA) (Tsutsui *et al.* 1997) to divide up the space. According to this idea, whenever the main population in P_{search} find a new optimum, it creates a new population in P_{track} and assign this new population to the optimum. To separate the sub-populations, Branke *et al.* (2000) provided each sub-population with a boundary containing all individuals of the population. This boundary is a hyper-cube determined by a centre (the most fit individual in the population) and a pre-defined range. To make sure that all individuals are inside the boundary, each sub-population is equipped with a different mutation step size relevant to the boundary. If an individual of one sub-population ventures to the area monitored by another sub-population, this individual will simply be discarded and re-initialised (this process is called *exclusion*). The same forking approach is also used in other EAs, for example DE (Mendes & Mohais 2005), (Lung & Dumitrescu 2007) . Similar approaches are also used in PSO. For example, in Multi-swarm charged PSO (mCPSO) (Blackwell 2007), swarms are also divided into sub-swarm in the same way as in SOS so that each swarm watches a different peak. In addition, mCPSO also maintain a similar mechanism (named anti-convergence) to the P_{search} in SOS so that there is always one free swarm to continue exploring the search space. Another example is

in Speciation PSO (Li *et al.* 2006) where each species is a hyper-sphere whose the centre is the best-fit individual in the species. Each species can be used to track a peak. To construct and separate species, periodically particles are regrouped according to their distance to each other. For clustering approaches, it is not always necessary to choose the best solutions as the centres of the clusters. In recent approaches (Li & Yang 2009, Woldesenbet & Yen 2009), density-based clustering methods are also used to divide/separate the sub-populations and to allow the algorithms explore different parts of the search landscape . It was reported that these density-based clustering techniques do help to improve the performance, but at the expense of additional computational cost to calculate the pair-wise distance among particles. The clustering-based approach is still widely used in recent EDO studies, for example in (Cheng & Yang 2010) to optimise the dynamic network routing problems.

The second approach is to incorporate some mechanism of penalty/rewarding to keep the sub-populations apart, of which SBGA Oppacher & Wineberg (1999) is a typical example. SBGA maintains the separation of populations by selecting individuals in P_{search} for reproduction according to their distance from the core in P_{track} rather than according to their original fitness values. The further an individual is from the core, the more likely that it will be reproduced.

The third approach is to estimate the basins of attractions of peaks and use these basins as the separate regions for each sub-population. MGA (Ursem 2000) is the first work following this approach. The authors provided a mechanism called *hill-valley detection*: given two individuals in the search spaces, they calculate the fitness of several random samples on the line between these two individuals. If the fitness in a sample point is lower than that of the two individuals, then a valley is detected. If a sub-population contains more than one valley, it will be split. A similar idea was implemented in a technical report in (Nguyen 2008*b*). Here firstly the basin of attraction of each peak is estimated using some simple sampling methods, then the search space is divided into hypercubes using a binary tree structure (KD-tree) in which each hypercube approximately covers the estimated basin-of-attractions. Each sub-population is then assigned to a hypercube. When a part of the landscape changes, the node covering the changing peak will adjust its structure and its hypercubes to make sure that the sub-populations are not overlapped. The advantage is the low computational cost (O(log N)) to identify where an individual belongs to. Initial results on the scenario 2 of the MPB benchmark (Branke 2001*b*) show that the proposed method can provide equal or better results than state-of-the-art methods.

Strengths and weaknesses

Methods following the multi-population approach have the following advantages:

1. Can maintain enough diversity for the algorithm to adaptively start a new search whenever a new change appears. Examples can be seen in the experiments in (Branke 2001*b*) where the proposed multi-population algorithm (SOS) was able to cover most of the peaks if given enough time while the non-multi-population GA could not.

2. Able to recall some information from the previous generation thanks to one (or several) population(s) dedicated for tracking and retaining previous solutions. This makes multi-population approaches usable in solving certain recurrent dynamic problems. For example, Ursem (2000) and Branke (1999) showed that the multi-population MGA and memory-based EA were able to recall good old solutions to deal with recurrent problems and hence outperformed normal EAs.

3. Can search/ track the moves of multiple optima, as analysed in many existing studies on multi-population, for example (Ursem 2000) and (Branke 2001*b*).

4. Can be very effective for solving problems with competing peaks or multimodal problems. (A survey of Moser (2007) showed that among 19 surveyed algorithms that are designed to solve the multimodal competing peaks benchmark Moving Peaks, a majory (15 out of 19) follow the multi-population approach).

The multi-population approach also has some disadvantages. They are:

1. Too many sub-populations may slow down the search. For example, Blackwell & Branke (2006) showed that for their multi-swarm PSO algorithm, if the number of sub-populations (swarms) is larger than the number of peaks, the performance of the algorithm decreases.

2. The need of calculating the distance/similarity/regional metrics to separate the sub-populations might also affect the performance. This cost has not been taken into account in most existing studies on multi-population approaches.

3. Lack of efficient memory for recurrent changes (keeping multiple populations does not guarantee an effective memory)

2.1.8 Summary about the strengths and weaknesses of current EAs for dynamic optimisation

From the literature review above we can conclude that each EDO approach seems to be suitable only for certain type of problem, which conforms to the No Free Lunch theorem (Wolpert & Macready 1997). The fact that each approach is likely to be suitable to some particular classes of problems is also the reason why many recent studies try to combine different approaches into one single algorithm to solve the problems better. The survey also shows that most existing methods were tested and evaluated only on academic problems. The question then is to find out (i) what are the common characteristics of existing academic problems; (ii) what are the common criteria to evaluate EDO algorithms; and more importantly (iii) whether these common characteristics and evaluation criteria reflect the common situations in real-world scenarios. In the next sections further investigations will be made to find the answers for these questions.

2.2 Performance measures

Properly measuring the performance of algorithms is vital in EDO. In this section I will (i) review existing studies to identify the most common criteria used to evaluate EDO algorithms, (ii) analyse the strengths and weaknesses of each measure, and (iii) discuss the possibility to improve the disadvantages (if there are any) of current performance measures. Performance measures in EDO can be classified into two main groups: optimality-based and behaviour-based. The subsections below will discuss each groups of measures in details.

2.2.1 Optimality-based performance measures

Optimality-based performance measures are measures that evaluate the ability of algorithms in finding the solutions with the best objective/fitness values (fitness-based measures) or finding the solutions that are closest to the global optimum (distance-based measures). This type of measure is by far the most common in EDO. The measures can be categorised into groups as follow:

Best-of-generation

This measure is calculated as the averages for many runs of the best values at each generation on the same problem. This performance measures is usually used in two ways: First, the best value in each generation is plotted against the time axis to create a performance curve.

This measure has been used since the early research in (Cobb 1990, Grefenstette 1992) as the *best-of-generation / best of population* in each generation and then as the *best objective value* (Bäck 1998), *best fitness* in each generation (Gaspar & Collard 1999) and *best-of-generation* (BOG) (Grefenstette 1999). This measure is still one of the most commonly used measures in the literature. The advantage of such performance curves is that they can show the whole picture of how the tested algorithm has performed. However, because the performance curve is not quantitative, it is difficult to compare the final outcome of different algorithms and to see if the difference between two algorithms is statistically significant (Morrison 2003).

To improve the above disadvantage, a variation of the measure is proposed where the BOG values is averaged over all generations (Yang & Yao 2003). The measure is described below:

$$\overline{F}_{BOG} = \frac{1}{G} \times \sum_{i=1}^{i=G} \left(\frac{1}{N} \times \sum_{j=1}^{j=N} F_{BOG_{ij}} \right) \tag{2.1}$$

where $\overline{F}_{BOG}$ is the mean best-of-generation fitness, G is the number of generations, N is the total number of runs, and $F_{BOG_{ij}}$ is the best-of-generation fitness of generation i of run j of an algorithm on a particular problem.

An identical measure to the $\overline{F}_{BOG}$, but with a different name, the *collective mean fitness*

$$F_C = \frac{1}{N} \times \sum_{j=1}^{j=N} \left(\frac{1}{G} \times \sum_{i=1}^{i=G} F_{BOG_{ij}} \right) \tag{2.2}$$

was proposed by Morrison (2003)at the same time. Morrison (2003) emphasized that the *collective mean fitness* should be calculated based on a sufficiently large number of generations to ensure that the final score is representative.

Recently the idea of calculating $\overline{F}_{BOG}$ and using F_{BOG} to plot performance curves was adapted in (Alba & Sarasola 2010*a*) to create two measures: the *area below a curve*, which is calculated as the definite integral of F_{BOG} (or other measures such as F_C or offline error/performance) over the optimisation process; and the *area between curves*, which is the area spanned between the performance curves of two algorithms.

The $\overline{F}_{BOG}$ is one of the most commonly used measures. The advantage of this measure, as mentioned above, is to enable algorithm designers to quantitatively compare the performance of algorithms. The disadvantage of the measure and its variants is that they are not normalised,

hence can be biased by the difference of the fitness landscapes at different periods of change. For example, if at a certain period of change the overall fitness values of the landscape is particularly higher than those at other periods of changes, or if an algorithm is able to get particular high fitness value at a certain period of change, the final $\overline{F}_{BOG}$ or F_C might be biased toward the high fitness values in this particular period and hence might not correctly reflect the overall performance of the algorithm.

Best-error-before-change

Proposed in (Trojanowski & Michalewicz 1999) and named *Accuracy* by the authors, this measure is calculated as the average of the best errors (the difference between the optimum value and the value of the best individual) achieved at the end of each change period (right before the moment of change).

$$E_B = \frac{1}{m} \sum_{i=1}^{m} e_B(i) \tag{2.3}$$

where $e_B(i)$ is the best error just before the ith change happens; m is the number of changes.

This measure is useful in situations where we are interested in the final solution that the algorithm achieved before the change. The measure also makes it possible to compare the final outcome of different algorithms. However, the measure also has three important disadvantages. First, it does not say anything about what the algorithms have done to achieve the current performance. Using this measure all algorithms that have the same final solutions before change will have the same score, regardless of how quick an algorithm recover from the last change and how fast it approaches the global solution. As a result, the measure is not suitable if what users are interested in is the overall performance or behaviours of the algorithms. Second, similar to the best-of-generation measure, this measure is also not normalised and hence can be biased toward periods where the errors are relatively very large. Third, the measure requires that the global optimum value at each change is known. Fourth, the measure requires that the time a change occurs is known.

This measures is adapted as the basis for one of the complementary performance measures in the CEC'09 competition on dynamic optimisation (Li *et al.* 2008).

Modified offline error and offline performance

Proposed in (Branke 2001*b*) and (Branke & Schmeck 2003), the *modified offline error* is measured as the average over, at every evaluation, the error of the best solution found since the last change of the environment. This measure is always greater than or equal to zero and would be zero for a perfect performance.

$$E_{MO} = \frac{1}{n} \sum_{j=1}^{n} e_{MO}(j) \tag{2.4}$$

where n is the number of generations so far, and $e_{MO}(j)$ is the best error since the last change gained by the algorithm at the generation j.

A similar measure, the *modified offline performance*, is also proposed in the same reference to evaluate algorithm performance in case the exact values of the global optima are not known

$$P_{MO} = \frac{1}{n} \sum_{j=1}^{n} F_{MO}(j) \tag{2.5}$$

where n is the number of generations so far, and $F_{MO}(j)$ is the best performance since the last change gained by the algorithm at the generation j.

With this type of measure, the faster the algorithm to find a good solution, the higher the score. Similar to the $\overline{F}_{BOG}$, the offline error/performance are also useful in evaluating the overall performance of an algorithm and to compare the final outcomes of different algorithms. These measures however have some disadvantages. First, it requires that the time a change occurs is known. Second, similar to $\overline{F}_{BOG}$, these measures are also not normalised and hence can be biased under certain circumstances.

Recently based on the modified offline error a new measure named *best known peak error* (BKPE) (Bird & Li 2007) was proposed to measure the convergence speed of the algorithm in tracking optima. The BKPE is calculated to every known peak. Immediately before a change, the error of the best individual on a known peak is added to the total error for the run. Similar to the offline error, after a change all current peak errors are reset to zero.

Relative-ratio-of-best-value

In the technical reports which defines the bechmark problems for the CEC'09 competition on dynamic optimisation (Li *et al.* 2008), a new performance measure was proposed based on the relative ratio between the best value gained by an algorithm and the global optimum value. The

performance is calculated as follows:

$$\begin{aligned} \text{performance} &= \sum_{i=1}^{\text{runs}} \sum_{j=1}^{num\_change} r_{ij} / \left(num\_change \times \text{runs} \right); \\ r_{ij} &= r_{ij}^{last} / \left(1 + \sum_{s=1}^{S} \left(1 - r_{ij}^{s} \right) / S \right) \\ r_{ij}^{s} &= f_j \left(x_{best} \left(s \right) \right) / f_j \left(x^* \right) \\ r_{ij}^{last} &= f \left(x_{best} \left(last \right) \right) / f_j \left(x^* \right) \end{aligned}$$

where S is the total number of sampling steps at each change period, $f_j \left(x_{best} \left(s \right) \right)$ is the best value the algorithm achieves at the s-th sampling step of the j-th change period, $f_j \left(x^* \right)$ is the global optimum value at the j-th change period, and *last* is the moment just before the next change happens.

This measure is similar to the measures E_{MO} and F_{BOG} in the way it takes into account the overall optimisation process and rewards algorithms that recover more quickly and hence it also has the same advantages and disadvantages as the other two measures.

Optimisation accuracy

The *optimisation accuracy* measure (also known as the *relative error*) was initially proposed in (Feng *et al.* 1997) and was adopted in (Weicker 2002) for the dynamic case:

$$accuracy_{F,EA}^{(t)} = \frac{F(best_{EA}^{(t)}) - Min_F^{(t)}}{Max_F^{(t)} - Min_F^{(t)}} \tag{2.6}$$

where $best_{EA}^{(t)}$ is the best solution in the population at time t, $Max_F^{(t)} \in \mathbb{M}$ is the best fitness value of the search space and $Min_F^{(t)} \in \mathbb{M}$ is the worst fitness value of the search space. The range of the accuracy measure ranges from 0 to 1, with a value of 1 and 0 represents the best and worst possible values, respectively.

The optimisation accuracy have the same advantages as the $\overline{F}_{BOG}$ and E_{MO} in providing quantitative value and in evaluating the overall performance of algorithms. The measure has an advantage over $\overline{F}_{BOG}$ and E_{MO}: it is independent to fitness rescalings and hence become less biased to those change periods where the difference in fitness becomes particularly large. The measure, however, has a disadvantage: it requires information about the absolute best and worst fitness values in the search space, which might not always be available in practical situations. In

addition, as pointed by the author himself (Weicker 2002), the optimisation accuracy measure is only well-defined if the complete search space is not a plateau at any generation t, because otherwise the denominator of Eq.2.6 at t would be equal to zero.

Distance-based measures

Although most of the optimality-based measures are fitness-based, some performance measures do rely on the distances from the current solutions to the global optimum to evaluate algorithm performance. In (Weicker & Weicker 1999), a performance measure, which is calculated as the minimum distance from the individuals in the population to the global optimum, was proposed. In (Salomon & Eggenberger 1997), another distance-based measure was introduced. This measure is calculated as the distance from the mass centre of the population to the global optimum.

Euclidean distance-based measures are also commonly used to evaluate the performance of dynamic multi-objective (DMO) optimisation algorithms. In (Zeng *et al.* 2006) the performance of DMO algorithms are evaluated based on the *generational distance* (GD) (Van Veldhuizen 1999) between the approximated front (which contains the current best function values) and the Pareto optimal front at the moment just before a change occurs. In (Farina *et al.* 2004) two measures, one is based on the minimum Euclidean distance between members of the approximated front and the Pareto front, and the other is based on the minimum Euclidean distance between members of the approximated set and the Pareto set, were proposed. In (Hatzakis & Wallace 2006), these two measures were extended using the idea of modified offline-error. In (Li *et al.* 2007), a modified version of the original GD named *reversed GD* was proposed for the dynamic case. The reversed GD is different from the dynamic GD in (Zeng *et al.* 2006) in that the distance between the Pareto front and the approximated front is calculated in a "reversed" direction, i.e. the calculation starts from each sampling point in the Pareto front and then find the closest solution in the approximated front, not the other way round as usual. In (Goh & Tan 2009*a*), an offline measure named *variable space generational distance* was also proposed and was calculated based on the distance between the approximated set and the Pareto set at each time step.

The advantage of distance-based measures is that they are independent to fitness rescalings and hence are less affected by possible biases caused by the difference in fitness of the landscapes

in different change periods. The disadvantages of these measures are that they require knowledge about the exact position of the global optimum, which is not always available in practical situation. In addition, compared to some other measures this type of measure might not always correctly approximate the exact adaptation characteristics of the algorithm under evaluated, as shown in an analysis in (Weicker 2002).

2.2.2 Behaviour-based performance measures

Behaviour-based performance measures are those that evaluate whether EDO algorithms exhibit certain behaviours that are believed to be useful in dynamic environments. Example of such behaviours are maintaining high diversity through out the run; quickly recovering from a drop in performance when a change happens, and limiting the fitness drops when changes happen. These measures are usually used complementarily with optimality-based measures to study the behaviour of algorithms. They can be categorised into the following groups:

Diversity

Diversity-based measures, as their name imply, are used to evaluate the ability of algorithms in maintaining diversity to deal with environmental dynamics. There are many diversity-based measures, e.g. *entropy* (Mori *et al.* 1997), *Hamming distance* (Oppacher & Wineberg 1999, Rand & Riolo 2005*a*, Yang 2008), *moment-of-inertia* (Morrison & De Jong 2002), *peak cover* (Branke 2001*b*), and *maximum spread* (Goh & Tan 2009*a*) of which Hamming distance-based measures are the most common.

Hamming distance-based measures for diversity have been widely used in static evolutionary optimisation and one of the first EDO research to use this measure for dynamic environments is the study of (Oppacher & Wineberg 1999) where the *all possible pair-wise Hamming distance* among all individuals of the population was used as the diversity measure. In (Rand & Riolo 2005*a*) the measure was modified so that only the Hamming distances among the best individuals are taken into account.

A different and interesting diversity measure is the *moment-of-inertia* proposed by Morrison & De Jong (2002). This measure is inspired by the fact that in engineering problems, in case an object rotates around its *centroid* (centre of mass), the moment of inertia can be used to measure how far the mass of the object is distributed from the centroid. Morrison & De Jong (2002) applied this idea to measuring the diversity of EA population. Given a population of

P individuals in N-dimensional space, the coordinates $C = (c_1, ..., c_N)$ of the centroid of the population can be computed as follows:

$$c_i = \frac{\sum_{j=1}^{P} x_{ij}}{P}$$

where x_{ij} is the ith coordinate of the j individual and c_i is the ith coordinate of the centroid.

Given the computed centroid above, the measure of diversity, named moment-of-inertia, of the population is:

$$I = \sum_{i=1}^{N} \sum_{j=1}^{P} (x_{ij} - c_i)^2$$

In (Morrison & De Jong 2002), the authors proved that the moment-of-inertia measure is equal to the pair-wise Hamming distance measure. The moment-of-inertia, however, has an advantage over the Hamming distance measure: it is more computationally efficient. The complexity of computing the moment-of-inertia is only linear with the population size P while the complexity of the pair-wise diversity computation is quadratic.

Another interesting, but less common diversity measure is the *peak cover* (Branke 2001*b*), which counts the number of peaks covered by the algorithms over all peaks. This measure requires full information about the peaks in the landscape and hence is only suitable in academic environment.

Diversity measures are also used in dynamic multi-objective approaches. In (Goh & Tan 2009*a*) the *maximum spread* commonly used in static MO was modified for the dynamic case by calculating the average value of the maximum spread over all generations as time goes by. In (Li *et al.* 2007), the diversity-based *hypervolume* (HV) measure (Van Veldhuizen 1999) commonly used in static MO was extended to a dynamic measure HVR(t), which is the ratio between the dynamic HV of the approximated front and the Pareto front.

Drops in performance after changes

Some EDO studies also develop measures to evaluate the ability of algorithms in restricting the drop of fitness when a change occurs. Of which, the most representative measures are the measures *stability* (Weicker 2002), *satisficability* and *robustness* (Rand & Riolo 2005*a*).

The measure *stability* is evaluated by calculating the difference in the fitness-based *accuracy*

measure (see Eq. 2.6) of the considered algorithm between each two time steps

$$stab^{(t)}_{F,EA} = max\{0, accuracy^{(t-1)}_{F,EA} - accuracy^{(t)}_{F,EA}\} \tag{2.7}$$

where $accuracy^{(t)}_{F,EA}$ has already been defined in Eq. 2.6.

The *robustness* measure is similar to the measure *stability* in that it also determines how much the fitness of the next generation of the GA can drop, given the current generation's fitness. The measure is calculated as the ratio of the fitness values of the best solutions (or the average fitness of the population) between each two consecutive generations.

The *satisficability* measure focuses on a slightly different aspect. It determines how well the system is in maintaining a certain level of fitness and not dropping below a pre-set threshold. The measure is calculated by counting how many times the algorithm is able to exceed a given threshold in fitness value.

Convergence speed after changes

Convergence speed after changes, or the ability of the algorithm to recover quickly after a change, is also an aspect that attracts the attention of various studies in EDO. In fact many of the optimality-based measures, such as the offline error/performance, best-of-generation, relative-ratio-of-best-value discussed previously can be used to indirectly evaluate the convergence speed. In addition, in (Weicker 2002) the author also proposed a measure dedicated to evaluating the ability of an adaptive algorithm to react quickly to changes. The measure is named *reactivity* and is defined as follows:

$$react^{(t)}_{F,A,\epsilon} = min\left\{ t' - t | t < t' \leqslant maxgen, t' \in \mathbb{N}, \frac{accuracy^{(t')}_{F,A}}{accuracy^{(t)}_{F,A}} \geqslant (1-\epsilon) \right\} \cup \{maxgen - t\} \tag{2.8}$$

where *maxgen* is the number of generations. It should be noted that this measure is only meaningful if there is actually a drop in performance when a change occurs. Otherwise, the value of the measure *reactivity* is always zero and nothing can be said about how well the algorithm reacts to changes. In situations like the dynamic constrained benchmark problems in (Nguyen & Yao 2009*a*) where the total fitness level of the search space increases after a change, the measure *reactivity* cannot be used.

Fitness degradation over time

A recent experimental observation (Alba & Sarasola 2010*b*) showed that in dynamic optimisation environments the performance of an algorithm might degrade over time due to the fact that the algorithm fails to follow the optima after some changes have occured. To measure how an algorithm degrades as the search advances due to the above reason, in (Alba & Sarasola 2010*b*) a measure named $\beta-degradation$ was proposed. The measure is calculated by firstly using linear regression (over the accuracy values achieved at each change period) to create a regression line, then evaluate the measure as the slope of the regression line. A positive $\beta-degradation$ value might indicate that the algorithm is able to keep track with the moving optima. This measure is among the first to consider the impact of tracking-performance degradation over the long term in DO. The measure however does not indicate whether the degradation in performance is really caused by the long-term impact of DOP, or simply by an increase in the difficulty level of the problem after a change. In addition, a positive $\beta-degradation$ value might also not always an indication that the algorithm is able to keep track with the moving optima. In problems where the total fitness level increases, like in the dynamic constrained benchmark problems in (Nguyen & Yao 2009*a*) mentioned above, a positive $\beta-degradation$ can be achieved even when the algorithm stays at the same place.

Robustness over time

Recently Yu *et al.* (2010) discussed some interesting evaluation criteria to measure the ability of algorithms in finding optimal solutions that are *robust over time*. According to Yu *et al.* (2010) a solution is called robust over time when it is used for at least two consecutive changing periods. When the solution quality becomes unsatisfactory, a new robust solution must be found. Some evaluation criteria to evaluate robust over time were suggested based on the quality of a solution from: (i) the perspective of a single solution; (ii) the perspective of the whole sequence of solution; and (iii) the perspective of the search efficiency of the algorithm. However, no experimental examples were provided to show how the criteria can be applied to a concrete problem.

2.2.3 Discussion

There are some open questions about performance measures in EDO. First, it is not clear if optimality is the only goal of real-world DOPs and if existing performance measures really reflect what practitioners would expect from optimisation algorithms. So far only a few studies e.g. (Rand & Riolo 2005*a*, Yu *et al.* 2010) tried to justify the meaning of the measures by suggesting some possible real-world examples where the measures can be applicable. It would be interesting to find the answer for the question of what are the main goals of real-world DOPs, how existing performance measures reflect these goals and from that investigate if it is possible to make the performance measures to be more specific (if needed) to suit practical requirements. In Chapter 3 an attempt will be made to find out more about the main optimisation goals of real-world DOPs and the link between existing performance measures and the goals of real-world applications.

Second, as shown in the literature review in this section, many optimality-based measures are not normalised and hence might be biased by fitness rescalings and other disproportionate factors caused by the changing landscapes. The *accuracy* measure (Weicker 2002) is among the few studies that tried to overcome this disadvantage by normalising the fitness values at each change period using a window of the maximum and minimum possible values. This approach however requires full knowledge of the maximum and minimum possible values at each change period, which might not be available in practical situations. In Subsection 6.4.1, a new measure, the *normalised score*, will be provided to facilitate comparing the performance of algorithms in a normalised way without using problem-specific knowledge.

Third, although the behaviour-based measures are usually used complementary with the optimality-based measures, it is not clear if the former really correlate with the latter. Recent studies(Alba & Sarasola 2010*b*) have shown that the behaviour-based measure *stability* does not directly relate to the quality of solutions and the results of the behaviour-based measure *reactivity* are "usually insignificant" (Alba *et al.* 2007, Alba & Sarasola 2010*b*). It would be interesting to systematically study the relationship between behaviour-based measures and optimality-based measures, and more importantly the relationship between the quality of solutions and the assumptions of the community about the expected behaviours of DO algorithms.

2.3 Benchmark problems

2.3.1 Properties of a good benchmark problem

The use of benchmark problems is crucial in the process of developing, evaluating, and comparing EDO algorithms. According to (Branke 2001*b*, Yang 2004, Morrison 2004, Younes 2006), a good benchmark problem is one that has the following characteristics:

1. Flexibility: Configurable under different dynamic settings (change severity, frequency, periodicity) and different scales (number of optima, dimensions, domain ranges etc)
2. Simplicity and efficiency: Simple to implement/analyse/evaluate and computationally efficient
3. Generalisation: Possible to represent different schenarios or different types of problems. In other words, the benchmark problem should not be very specific.

In addition, because the ultimate goal of any optimisation algorithm is to be applicable to real-world situations, a good benchmark problem needs to satisfy the following important property:

4. Allow conjectures to real-world problems or resemble real-world problems to some extents. (Branke 2001*b*, Goh & Tan 2009*b*)

2.3.2 Reviewing existing general-purpose benchmark generators/problems

In this section, I will review the commonly used general-purpose dynamic optimisation benchmark generators/problems in the literature based on the above criteria. The purpose is to identify the common characteristics of benchmark problems, and from that in the next chapter we will see if these characteristics reflect the properties of real-world problems.

When reviewing existing benchmark generators/problems, we can either categorise problems based on the ways they are generated, or based on the characteristics of the generated problems. In this section I choose the second way of categorisation because (i) it better suits the purpose of identifying the common characteristics of benchmark problems and (ii) it helps users in choosing the suitable benchmark for their applications. In the end what users look for in selecting a benchmark problem is not how they are generated but what types of dynamics they represent and what characteristics they have.

The characteristics of each general-purpose benchmark generator/problem are identified and the problems are classified into different groups based on the following different criteria:

1. Time-linkage: Whether the future behaviour of the problem depends on the current solution found by the algorithm or not.
2. Predictability: Whether the generated changes are predictable or not
3. Visibility: Whether the changes are visible to the optimisation algorithm and if so whether changes can be detectable by using just a few detectors
4. Constrained problem: Whether the problem is constrained or not
5. Single/multiple objective
6. Type of changes: Detailed explanation of how changes occur in the search space
7. Changes are cyclic/periodical/recurrent or not?
8. Factors that change: Parameter of objective functions / Domain of variables / Number of variables / Constraints / Other parameters

Tables 2.1 and 2.2 provides the detailed information of each benchmark problem in the continuous and combinatorial domains, respectively, and their characteristics.

2.3.3 The common characteristics of existing benchmark generators/problems

From tables 2.1 and 2.2, we can see that the common characteristics of academic benchmark problems are as follow:

- *All of the reviewed general-purpose benchmark generators/problems are non time-linkage problems.* There are a couple of general-purpose benchmark problems with the time-linkage property (Bosman 2005, Nguyen & Yao 2009*b*), but they are proposed as a proof of principle rather than a complete set of benchmark problems.
- *Most of the reviewed benchmark generators/problems are unconstrained or domain constrained*, except the two most recent studies (Nguyen & Yao 2009*a*, Richter 2010)

- *In the default settings of most of the review benchmark generators/problems, changes are detectable by using just a few detectors.* Exceptions are some problem instances in (Cobb & Grefenstette 1993, Trojanowski & Michalewicz 1999) where only one or some peaks move, and in (Weicker 2000, Nguyen & Yao 2009*a*, Richter 2010) where the presences of the visibility mask or constraints make only some parts of the landscapes change. Due to their highly configurable property some benchmark generators can be configured to create scenarios where changes are more difficult to detect.

- *In most cases the factors that change are the objective functions.* Exceptions are one instance in (Li *et al.* 2008) where the dimension also changes and the problems in (Nguyen & Yao 2009*a*, Richter 2010) where the constraints also change.

- *Many generators/problems have unpredictable changes in their default settings*, but due to their flexibility some of the generators/problems can be configured to allow predictable changes, at least in the frequency and periodicity of changes

- *A majority of benchmark generators/problems have cyclic/periodical/recurrent changes*

- *Most generators/problems are single-objective* except the problems in (Jin & Sendhoff 2004) and (Farina *et al.* 2004). Recently there are some new dynamic multi-objective problems e.g. (Zhou *et al.* 2007), but most of them are based on the two papers mentioned above.

The common characteristics of academic benchmark problems above reflect the current main assumptions of the EDO community about the characteristics of DOPs. In the next chapter we will identify if these characteristics are also common in real-world applications and if there is any characteristic that has not been covered in existing research.

Table 2.1: Common general-purpose benchmark generators/problems in the continuous domain

	General notes	Time-linkage	Changes are predictable?	Changes are detectable by using just a few detectors?	Constr problem (domain constr not count)?	Single/ Multi Obj?	Type of changes	Changes are cyclic/ periodical/ recurrent?	Factors that change: Objective functions	Domain of variables	Number of variables	Constr. functions	Others notes
Switching function (Cobb & Grefenstette 1993)	The benchmark consists of two landscapes A and B. Changes can occur in three ways: (1) linear translation of peaks in A ; (2) global optimum randomly moves while the rest of landscape A is fixed; (3) switching landscapes between A and B.	No	Mostly no (for the type of changes where peaks are linearly translated, peak movements might be predictable)	Yes & No (There are three types of changes. The first and the last can be detected by using a few detectors, while the second type of change is not)	No	Single-objective	Three types of changes: (1) linear translation of peaks; (2) global optimum randomly moves while the landscape is fixed; (3) switching landscapes.	Yes, in both fast (2 generations) and slow (20 generations) modes	N/I (no detail of the objective function is given)	No	No	No	Linear translation of all peaks; random movement of global opimum and switching landscape
Moving Peaks (Branke 1999)	The search landscape consists of a number of randomly generated peaks, each has its width, height and location changed after each change step. The benchmark is highly configurable (dimension, number of peaks and the dynamics of each peak are all configurable)	No	Mostly no in the default settings but some factors can be predictable if specifically configured (e.g. where the parameter lambda=1, the peaks move in the same direction and hence movement direction is predictable)	Yes in the default settings but can be configurable (the benchmark generator can be modifiable to allow changes in only a part of the landscape to make changes more difficult to detect)	No	Single-objective	Changes in heights, widths and locations of peaks. The widths and heights of peaks are changed by adding a Gaussian variable. The location of peaks are moved by a fixed step and the direction of peaks are based on a combination of the previous direction and a direction parameter.	No but configurable	Yes	No	No	No	Each of the peaks has its own time-dependent parameters height, width and location and hence each peak can change differently. E.g. Yu *et al.* (2010) configured the benchmark to make different peaks change with different frequencies and severities.
Oscillating Peaks (Branke 1999)	The search landscape oscillates among L fixed landscapes.	No	No (it might be possible to predict the period of oscillation)	Yes	No	Single-objective	Landscape switching	Yes (due to the oscillation of landscapes)	No	No	No	No	Landscape switching
DF1 (Morrison & DeJong 1999, Morrison 2003)	The search landscape consists of a number of peaks (randomly generated or pre-determined), each has its width, height and location changed after each time step. The behaviours of changes are controlled by a logistic function. The benchmark is highly configurable (dimension, number of peaks and the dynamics of each peak are all configurable)	No	No in the five tested instances provided in (Morrison 2003) but some factors can be predictable if specifically configured (e.g. where the motion of peaks is set to be linear, peaks' movement directions can be predictable)	Yes in four tested instances, no in one test instance and configurable (the benchmark generator can be modifiable to allow changes in only a part of the landscape to make changes more difficult)	No	Single-objective	Changes in heights, widths and locations of peaks. The behaviours of changes are controlled by a logistic function. Depending on the parameter of the logistic function, changes in step sizes can be fixed, birfurcation or chaotic.	No in the five tested instances provided in (Morrison 2003) but can be configurable	Yes	No	No	No	

Table 2.1 Continuous benchmark generators/problems (cont.)

	General notes	Time-linkage	Changes are predictable?	Changes are detectable by using just a few detectors?	Constr problem (domain constr not count)?	Single/ Multi Obj?	Type of changes	Changes are cyclic/ periodical/ recurrent?	Factors that change: Objective functions	Domain of variables	Number of variables	Constr. functions	Others notes
Gaussian peak (Grefenstette 1999)	The search landscape consists of a number of peaks (randomly generated), each has its location changed after each time step. Two levels of severity: abrupt and gradual, were tested	No	No (all peaks move randomly)	Yes	No	Single-objective	Changes in location of peaks. Peaks move in random directions and the step sizes are uniformly distributed over an interval controlled by the level of severity.	No	Yes	No	No	No	
Disjoint landscape (Trojanowski & Michalewicz 1999)	The main principle of this benchmark generator is to divide the search space into a number of disjoint sub-spaces, each with a separate unimodal function. The main search space hence is a composition of the local optima from the disjoint sub-spaces.	No	Mostly no but configurable (it is possible to configure the benchmark to make it predictable. In addition, in the tested example peaks' values are artificially move in a circle, making it to some extent possible to predict the movement)	Dependable on the number of peaks that change at each time step (in the tested example, only the values of some peaks change)	No	Single-objective	Changes in values of peaks	Yes	Yes	No	No	No	
Dynamic rotation (Weicker & Weicker 2000)	The principle of this benchmark generator is to combine the original search space with a "visibility mask", which allows only certain parts of the search space to have the original fitness values. Other regions, which are hidden by the mask, have constant, pre-defined fitness values. The dynamics are created by rotating the original search space, the mask, or both.	No	Mostly no (because all changes in this generator are created by rotation, to some extent we can consider the rotation movement predictable)	Partly (in case there is no visibility mask, it is possible to detect changes using just one detector. In case there is a visibility mask, the level of difficulty in detecting changes depends on the way the visibility mask is defined)	No	Single-objective	Rotations of the underlying search space and the visibility masks. The rotation is controlled by an orthogonal matrix.	Yes (due to the rotation)	Yes	No	No	No	Changes in visibility masks
MOO-based dynamic problem generator (Jin & Sendhoff 2004)	The principle of this benchmark generator is to use the aggregating objectives approach to create a n-objective dynamic function from n+1 static single-objective functions through a dynamic weight. The dynamic weight governs how the dynamic problem changes. The benchmark is highly configurable	No	Yes and Configurable (it is possible to configure the benchmark generator to create predictable changes. For example, in the tested instance the optimum movement is configured to be linear, and hence could be predictable)	Yes	No	Both single and multiple-objectives are configurable	The global optimum (or the Pareto front in the multiple-objective case) can be configured to move linearly, non-linearly or to follow specific moving rules. The height of the peak also changes accordingly.	Not in the tested instance but configurable	Yes	No	No	No	The dynamic parameter of the main objective function is the aggregate weight, which controls how the problem changes

Table 2.1 Continuous benchmark generators/problems (cont.)

	General notes	Time-linkage	Changes are predictable?	Changes are detectable by using just a few detectors?	Constr problem (domain constr not count)?	Single/ Multi Obj?	Type of changes	Changes are cyclic/ periodical/ recurrent?	Factors that change				
									Objective functions	Domain of variables	Number of variables	Constr. functions	Others notes
FDA dynamic multi-objective benchmark set (Farina *et al.* 2004) (extended to ZJZ in (Zhou *et al.* 2007))	The principle of this benchmark generator is to combine the static multiple-objective functions with the time-dependent parameters: F(t) to control the dynamics of the density of Pareto solutions, H(t) to control the dynamic of the shape of the Pareto front, and G(t) to control the dynamic shape of the Pareto optimal set.	No	Configurable (it is possible to configure the benchmark generator to create predictable changes. It might also be possible to predict the period of oscillation)	Yes	No	Mulitple-objective	The density of Pareto solutions, the shape of the Pareto front and the shape of the Pareto set change over time	Yes	Yes	No	No	No	Changes in the objective functions are controlled by three time-dependent parameters: F(t), G(t) and H(t). FDA1 was extended in (Zhou *et al.* 2007) to add nonlinear linkages between variables and to make PF dynamic.
Dynamic test functions (Tfaili *et al.* 2007)	This benchmark set follows a landscape-oriented approach where the dynamic test problems are specifically designed to represent different changes in landscape structure, in optima's positions, in optima's values etc. Changes can be linear or periodical.	No	Partly (the changes in some problems in the benchmark set follow predictable rules like moving linearly or occuring periodically)	Yes for most tested instances (exceptions are in problems like the OPoL where only the position of the global optimum changes)	No	Single-objective	Different types of changes are generated in different functions, e.g. changes in landscape structure, in optima's positions, in optima's values etc. Changes can be linearly or periodically.	Yes	Yes	No	No	No	
CDOPG (XOR-extension for continuous domain) (Tinos & Yang 2007)	The principle in this generator is to use an orthogonal transformation matrix to periodically rotate a static landscape to create dynamic instances (in a similar way to the XOR benchmark in the combinatorial domain). The properties of the fitness landscape is preserved after each change	No	No (but the periodicity of rotations can be predictable)	Yes	No	Single-objective	The fitness landscape is rotated. The magnitude of change is defined by the rotation angle.	Yes (due to the rotation)	Yes	No	No	No	Changes (rotations) are made on the decision variables. Specifically, before being evaluated each individual vector is moved (rotated) to a different position in the fitness landscape using an orthogonal matrix.

Table 2.1 Continuous benchmark generators/problems (cont.)

	General notes	Time-linkage	Changes are predictable?	Changes are detectable by using just a few detectors?	Constr problem (domain constr not count)?	Single/ Multi Obj?	Type of changes	Changes are cyclic/ periodical/ recurrent?	Factors that change: Objective functions	Domain of variables	Number of variables	Constr. functions	Others notes
CEC09 GDBG (Li *et al.* 2008)	This set of benchmark generators is a combination of existing ideas about landscape shifting (Suganthan *et al.* 2005), landscape rotation (Salomon 1996, Weicker & Weicker 2000, Suganthan *et al.* 2005, Tinos & Yang 2007), and using dynamic rules to control change steps (Morrison 2003)	No	No (but the periodicity of rotations can be predictable)	Yes	No	Single-objective	The fitness landscape is rotated and shifted. The magnitude of change is defined by the rotation angle.	Yes (due to the rotation)	Yes	No	Yes (for each proposed problem, there is one instance with changing dimension)	No	The landscape is rotated and the heights/widths of peaks are also changed. Rotation are made on decision variables. The magnitude of changes (angle of rotation) is determined by dynamic rules (small/ large/ chaotic/ random/ recurrent/ noisy).
G24 dynamic constrained benchmark set (Nguyen & Yao 2009*a*)	The principle of this benchmark generator is to make existing benchmark problems dynamic by replacing their static parameters with time-dependent aprameters. The benchmark supports dynamics in the constraint functions and the problems are organised in pairs, of which each pair has two almost identical problem, one with a special property and one with not.	No	Yes (changes follow predictable rules like linear movements and periodical movements)	No (there are situations when only a part of the landscape changes due to dynamic constraints. In such case it might not be easy to detect changes using a few detectors)	Yes	Single-objective	Combinations of changes in objective functions, changes in constraints and changes in both. Changes are linear and cyclic.	Yes	Yes	No	No	Yes	Changes (linear and cyclic) are made on the parameters of the objective functions and constraint functions
A dynamic constrained benchmark problem (Richter 2010)	The principle of this benchmark generator is to combine existing "field of cones on a zero plane" with dynamic norm-based constraints (with square/diamond/sphere-like shapes)	No	No in the proposed settings	No (there are situations when only a part of the landscape changes due to dynamic constraints. The author also proposed an unconstrained version (Richter 2009) where the level of detectability is adjustable)	Yes	Single-objective	Locations of peaks and constraints are changed following a a Gaussian variable with fixed mean and variance	Partly (changes are recurrent because they are generated using a Gaussian variable with fixed mean and variance)	Yes	No	No	Yes	

Table 2.2: Common general-purpose benchmark generators/problems in the combinatorial domain

	General notes	Time-linkage	Changes are predictable?	Changes are detectable by using just a few detectors?	Constr problem (domain constr not count)?	Single/ Multi Obj?	Type of changes	Changes are cyclic/ periodical/ recurrent?	Factors that change: Objective functions	Factors that change: Domain of variables	Factors that change: Number of variables	Factors that change: Constr functions	Others notes
Dynamic Match Fitness (Stanhope & Daida 1998)	This benchmark generator is based on the static bit-matching function (find a solution that matches a given string). The dynamics elements are introduced by changing the match-string.	No	No in the default settings but configurable to be cyclic and hence its periodicity can be predictable	Dependable on particular changes (number of bits changed and the location of changing bits)	No	Single-objective	Changes are introduced by changing a number of bits in the match-string	No (not considered in the tested instances but configurable)	Yes	No	No	No	
XOR (Yang & Yao 2003)	This benchmark generator has a special property: it can be combined with any static binary-coded problems to generate dynamic problems. Dynamic problems are generated by XOR-ing each individual with a special binary mask, which determines the magnitude of changes (in term of Hamming distances). Dynamic landscapes generated by the XOR operator have a special property that the landscape structure (and hence the distances among individuals and their fitness values) is preserved after each change.	No	No in the default settings but configurable to be cyclic and hence its periodicity can be predictable	Dependable on particular changes and on the underlying landscape	N/A (depend on the chosen static problem)	Single-objective	Changes are introduced by changing a number of bits in the binary mask, which will later be used to transform the position of individuals in the population. The severity level of changes is represented by the Hamming distance between the old and new binary mask.	Yes (the original version does not support cyclic changes, but an extended version was proposed in (Yang 2005*b*)	Yes	No	No	No	Changes are made on the vector of decision variables. Specifically, before being evaluated each individual vector is moved to a different position in the fitness landscape using the XOR operator and the binary mask. In other words, instead of moving the optimum, using the XOR operator the search population is moved after each change.
Dynamic DTF (Yang 2004)	This benchmark generator is based on the static Unitation and Trap functions. The static functions are made dynamic by making theirs static parameters time-dependent and by changing the scales of function values. The benchmark generator is highly configurable	No	No in the default settings but configurable to be cyclic and hence its periodicity can be predictable	No (there is no guarantee that using a few detectors can detect changes because due to the nature of the dynamic trap function, only a part of the search landscape changes)	No	Single-objective	Changes in optima height, size of basin and both optima height and basin size. Other advanced dynamic environments can also be constructed	Not considered in the tested instances but configurable	Yes	No	No	No	

2.4 Summary: the assumptions in EDO academic research

In this chapter we have reviewed and categorised existing EDO studies on the solving methods (Section 2.1), performance measures (Section 2.2), and benchmark problems (2.3). The reviews showed us the strengths and weaknesses of each method and more importantly identified the common assumptions of the community about the characteristics of DOPs, which can be summarised as follow:

- *Optimisation goals*: *Optimality is the primary goal or the only goal in a majority of academic EDO studies*, as evidently shown by the large number of optimality-based measures reviewed in Section 2.2. Some studies do pay attention to developing other complementary measures (e.g. the behaviour-based measures in Subsection 2.2.2), but these complementary measures mainly focus on analysing the behaviours of the algorithms rather than checking if the algorithms satisfy users requirements.

- *The time-linkage property*: *Non time-linkage (the algorithm does not influence the future dynamics) is the main focus of academic EDO research*, as evidently shown by the fact that all commonly used general-purpose benchmark problems are non-time-linkage.

- *Constraints*: *Unconstrained problems are the main focus of academic research*, especially in the continuous domain, as shown by the majority of academic benchmark problems.

- *Visibility and detectability of changes*: Current EDO methods assume that changes either are known or can be easily detected using a few detectors.

- *Factors that change*: The common factors that change in academic problems is the objective function.

- *Reason for tracking*: The main assumption is that the optima (local or global) after change is close to the optima (local or global) before change, as shown in a majority of benchmark problems (although in the Moving peaks (Branke 1999) and DF1 (Morrison & DeJong 1999) benchmarks the new global optima are not close to the previous global optima, they are still close to a previous local optima). Due to that, tracking is preferred to restarting.

- *Predictability*: The predictability of changes has increasingly attracted the attention of the community. However, the number of studies in this topic is relatively still small compared to the unpredictable case

- *Periodicity*: The periodicity of changes is a given assumption in many mainstream approaches as *memory* and *prediction*.

The literature review also shows that not many of the assumptions above are backed up by evidence from real-world applications. This leads to the question of whether these academic assumptions still hold in real-world DOPs and if yes then whether these assumptions are representative in real-world applications and in what type of applications do they hold. In an effort to answer this question, in the next chapter I will carry out a detailed review of the characteristics of real-world DOPs, from that I will identify the overlaps and gaps between academic EDO research and real-world DOPs.

Chapter 3

Identifying the characteristics of dynamic optimisation problems: from academic evolutionary research to real-world problems

3.1 Motivation and research questions

As shown in the literature review in Chapter 2, many current evolutionary dynamic optimisation (EDO) studies focus on academic problems where certain assumptions are given and certain characteristics are investigated. However, it is unclear of whether these academic assumptions still hold in real-world dynamic optimisation problems (DOPs) and whether the considered characteristics are representative in real-world applications.

The lack of a clear link between EDO academic research and real-world scenarios has lead to some criticisms on how realistic current academic problems are. Ursem *et al.* (2002) downplayed the importance of current academic benchmarks by stating that "no research has been conducted to thoroughly evaluate how well they reflect characteristic dynamics of real-world problems"; Branke *et al.* (2005) pointed out that "little has been done to characterize and understand the nature of a change in real-world problems"; Rohlfshagen & Yao (2008) criticised that "a large amount of effort is directed at an academic problem that may only have little relevance in the real world."; and in (Nguyen & Yao 2009*a*, Nguyen & Yao 2009*b*) we showed that there are

some classes of real-world problems whose characteristics have not been captured by existing academic research yet.

Criticisms from the cited references above motivated me to investigate the following research questions about the gaps between real-world problems and academic research:

- In which problems the current assumptions of EDO academic research about DOPs (as listed in Subsection 2.4) hold?
- Is there any real-world characteristic not covered in EDO academic research?
- Is there any type of problem that has not yet received attention from the EC community?
- The distribution/popularity of different types of real-world problems?
- Current approaches used to solve real-world problems? Which types of problems are solved by EAs and which are not?

I believe that answering the questions above is vital in bridging the gap between academic research and real-world applications. To contribute to the task of answering this question, in this chapter I will carry out a review of a set of recent real-world dynamic optimisation references. From the review I will investigate for the first time some insights about the link between academic EDO research and certain classes of real-world DOPs. Details of the review, and the corresponding investigations will be described in the next sections.

3.2 A survey of real-world problems with dynamic/uncertainty environments

3.2.1 Survey purpose and methods

To help answering the research questions mentioned in section 3.1 above we took a detailed survey of recent references (from different disciplines) published in English (mostly from 2006-2008) from Inspec / Compendex / GeoBase / Referex that have real-world problems with dynamic/uncertain/non-stationary environments and are solved using evolutionary algorithms or other stochastic / approximation optimisation techniques in an *online* way (i.e. we surveyed those problems that are DOPs). The goal of the survey is not to cover the characteristics of

all types of real-world dynamic optimisation problems, but only to summarise some representative characteristics of dynamic real-world problems that have been studied recently. To narrow down the search space to a manageable scope, the survey does not cover all sub-field/topics listed out in the reference database. References using exact methods and other deterministic / non-approximate methods; and many references relating to neural networks, fuzzy set, power systems and image processing are not considered due to the way the search syntax (to be shown below) has been chosen. The characteristics of each real-world problem are identified and the problems are classified into different groups based on the following different criteria:

1. Time-linkage: Whether the future behaviour of the problem depends on the current solution found by the algorithm or not. We recognise a real-world problem from a particular reference as having the time-linkage property if this property is mentioned as existing in the problem (by the authors of the corresponding reference), regardless of whether the time-linkage property is handled by the authors of the reference or not. There might be cases where it is not clear if the time-linkage property exists in a problem. In such cases the corresponding problems are categorised in the "no information" group.

2. Solved by EA or other meta-heuristics or not

3. Continuous/discrete: Whether the changes happen in continuous or discrete time

4. Predictability: Whether the changes are predictable or not

5. Visibility: Whether the changes are visible to the optimisation algorithm or the algorithm needs to detect changes or adapt with changes by itself and whether the changes only occur in a part of the search space

6. Frequency of changes: Whether the frequency of changes is fixed or variable

7. Constrained problem: Whether the problem is constrained or not

8. Single/multiple objective

9. Optimisation goals

10. Factors that change: Parameter of objective functions / Domain of variables / Number of variables / Constraints / Other parameters

11. Type of dynamics (e.g. recurrent/cyclic, linear, non-linear, distribution etc.)

12. Restart/Tracking: whether the restart or tracking (re-using previous knowledge) approach is chosen and why

13. Origin of real-world data

14. Types of applications and disciplines

Method: The survey was mostly based on a search made on Inspec / Compendex / GeoBase / Referex on 04/2008 and updated on 07/2010 with the following syntax:

> ((((((((((dynamic wn KY OR uncertain* wn KY OR non-stationary wn KY OR online wn KY OR predict* wn KY) AND (optim* wn KY OR evolution* wn KY)AND (algorithm wn KY OR method wn KY OR technique wn KY)) AND (({optimization} OR {algorithms} OR {genetic algorithms} OR {optimisation}) WN CV)) AND (({921.5} OR {c1180}) WN CL)) NOT (({computational fluid dynamics} OR {constraint theory} OR {finite element method} OR {neural networks} OR {convergence of numerical methods} OR {iterative methods} OR {linear matrix inequalities} OR {fuzzy control} OR {control system synthesis} OR {monte carlo methods} OR {error analysis} OR {sensitivity analysis} OR {fuzzy sets}) WN CV)) NOT (({dynamic programming} OR {decision making} OR {numerical methods} OR {graph theory} OR {approximation theory} OR {distributed computer systems} OR {trees (mathematics)} OR {integer programming} OR {fuzzy set theory} OR {neural nets} OR {matrix algebra}) WN CV)) NOT (({mobile robots} OR {embedded systems} OR {bandwidth} OR {computation theory}) WN CV)) NOT (({921} OR {912.2} OR {922.1} OR {721.1} OR {722.4} OR {716.1} OR {c1140z}) WN CL)) NOT (({723.2} OR {922.2} OR {701.1} OR {c1160} OR {714.2}) WN CL)) NOT ({computer simulation} WN CV)) AND ({english} WN LA))

It should be noted that in this survey we only consider papers that use real-world data or solve problems originating from real-world situations. In case a reference takes real-world data to do the simulation, we will only consider it if the simulation can show how the method works in a real-world situation, i.e. how it can run in real-time and deal with different changes and the

optimisation goals are set clear. References, e.g. (Kiselev & Alhajj 2008, Ko *et al.* 2008, Huang & Wu 2008, Chaer & Monzon 2008) that do use real-world data but do not provide enough information about the characteristics of the data; or how the dynamics affect the proposed method and how the proposed method deal with the dynamics, are not considered. Real-world references where the problems are solved offline or where there is not enough information of whether and how the proposed method can be applied online e.g. (Takagi *et al.* 2008, Gao & Sheng 2008) are also not considered. Especially, for references of control systems, we only consider those that really use real-world data from hardware/physical systems or have hardware/physical implementations because in many control systems the dynamics come from the errors of real-world equipments and disturbances. Benchmark problems, even if designed to simulate real-world applications, are not considered unless there is evidence that the data used to create the benchmark are taken from real-world applications. Because of that, many references that use such common benchmark problems (e.g. dynamic modifications of the Solomon set for VRPTW (Solomon 1987)) will not be considered.

It is also worth noting that in many selected references, e.g. (Deb *et al.* 2007, Soga *et al.* 2008, Wang, Wu & Liu 2008, Ahmad & Liu 2008, Ngo *et al.* 2006, Kanoh 2007), the real-world data is used only as a framework on which the authors test different artificial dynamic scenarios. In such case we will only consider the parts that evidently originate from real-world applications.

Such strict criteria in selecting references are needed because the purpose of the survey is not to list all real-world references that we found using the above syntax, but to investigate the characteristics of real-world problems and their effects on optimisation algorithms. Such strict criteria, however, certainly cannot cover all relevant real-world references in the period 2006-2008. First, the survey might omit many references (particularly the scheduling problems and control problems) that actually originate from real-world situations, or actually use real-world data just because the authors of these references do not explicitly state so in their papers. Second, the survey is not able to cover real-world situations from certain disciplines where it is prohibited to disclose the source of real-world data (e.g. in commercial applications) or where it is only possible to simulate real-world situations using simulations (e.g. in military applications or space applications). Third, the survey only focuses on references that satisfy the search syntax and the time frame described above. Fourth, although we did our best to judge the relevance of the filtered references, the selections and classifications only reflect our views of real-world

problems, which might not cover all types of applications that are of interest. However, even with such limitations, it is hoped that the outcome of the survey will still be useful to reveal for the first time the common characteristics of a set of representative real-world DOPs, and the links between these problems and academic EDO research.

From the 2394 filtered references, we selected the references that are most relevant according to the selection criteria above. We also looked at the list of papers cited by the selected references and read those that seem to be relevant. Eventually, 56 most relevant problems (which are problems that satisfy the definition of DOPs in Chapter 1 and have enough detailed information about their characteristics) are selected for a detailed survey.

To the best of our knowledge, there has been no similar survey on real-world DOP references to investigate the links between EDO academic research and real-world applications. The only related study is the work of Andrews & Tuson (2005). This research does not survey existing references in the literature, but reports questionnaires and interviews of the authors with four practitioners to investigate their views on the characteristics of some real-world applications that they have experienced. Although the research in (Andrews & Tuson 2005) does not directly solve any real-world DOPs, it does provide important information on some characteristics of the real-world problems that the four practitioners have worked with. Due to that, in this survey we will also include these reported characteristics in our classification.

Tables 1, 2, 3, 4 and 5 in the Appendix (page 246) list the references we selected with brief information about their characteristics based on the criteria above. The tables are divided into four groups: combinatorial applications solved by EAs and other meta-heuristics; continuous applications solved by EAs and other meta-heuristics; combinatorial applications solved by other methods; and continuous applications solved by other methods. These tables contain the most relevant details of the characteristics of the surveyed problems. For all information that I classified using the criteria above, please see the online report at `http://www.cs.bham.ac.uk/~txn/reports/classified_apps.pdf`.

In the next subsections I will summarise our findings from the survey on different aspects.

3.2.2 General observations

In this subsection I will summarise the general characteristics of the surveyed applications and how they match with existing EA academic research.

Distribution of applications

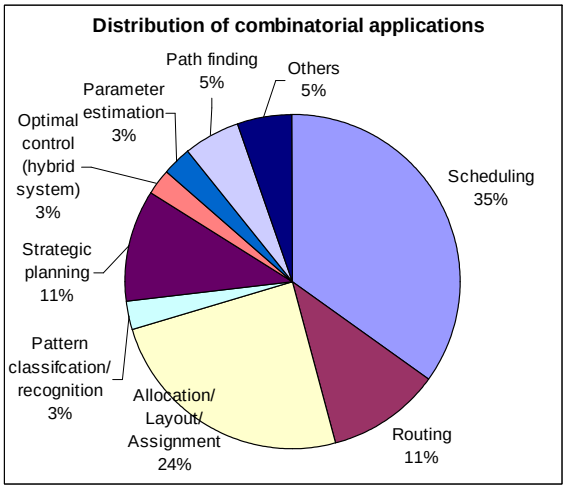

Figure 3.1: Distribution of 29 combinatorial applications in the surveyed references

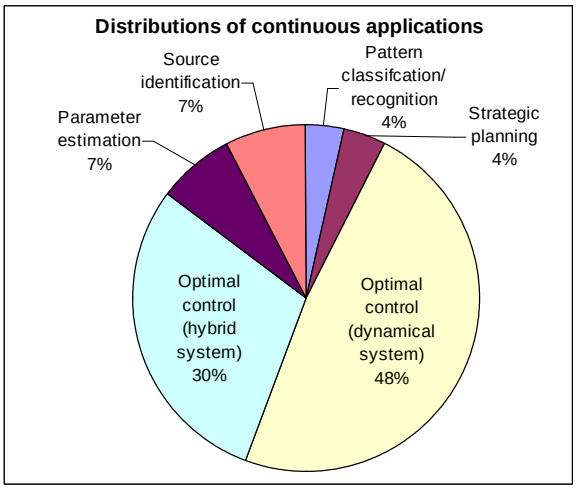

Figure 3.2: Distributions of 27 continuous applications in the surveyed references

The first aspect to be analysed is the distributions of different types of applications in the surveyed references, which hopefully will provide EDO researchers with a better view of what are the more common types of applications among real-world DOPs.

For this analysis the applications are grouped into two categories: Applications where the decision variables are in the combinatorial domain (Figure 3.1) and applications where the decision variables are in the continuous domain (Figure 3.2). It should be noted that there are real-world systems that have both continuous and combinatorial decision variables. Especially,

most hybrid systems that we surveyed have both continuous and combinatorial decision variables, and hence they can be classified into both the continuous applications and the combinatorial applications. In this survey such systems are classified into the continuous group due to their similarity to the continuous dynamical systems. Of course they can also be classified into a third different category.

The analysis results show that for the combinatorial domain, the most common types of DOPs are scheduling (35%), allocation/layout/assignment (24%), routing and planning (both 11%). They are also among the most common types of problems considered in EA combinatorial research. For applications in the continuous domain, a majority of the surveyed problems are optimal control problems (including normal dynamical systems (48%) and hybrid systems (30%)), which have not yet attracted much interest from the EA community. The other types of problems are parameter estimation, source identification, pattern classification and planning.

The coverage of EAs and other meta-heuristics

Because the focus of this chapter is on EDO research, it might be more of interest to investigate the coverage of EAs and other meta-heuristics on the surveyed applications. Analyses on the distribution of applications that are solved by EAs and other meta-heuristics in the combinatorial/continuous domains are given in Figures 3.3, 3.4, respectively, and analyses on the way EAs and other meta-heuristics are used to solve the applications in the combinatorial/continuous domains are given in Figures 3.5, and 3.6, respectively.

The results show that EA and other meta-heuristics are used to solve 36% of the surveyed applications, confirming the popularity of these methods in solving online applications. Among these applications, EAs/meta-heuristics are used in one of the following three ways:

- EAs/meta-heuristics are used as the main dynamic solvers to produce the dynamic solutions for the problem (24% in the combinatorial domain and 28% in the continuous domain)

- EAs/meta-heuristics are not the main/only dynamic solvers (10% in the combinatorial domain and 4% in the continuous domain). Instead they are used to optimise a part of the problem (e.g. in the airport-scheduling problem (Atkin *et al.* 2008), a meta-heuristic (Tabu search) is used for the initial search of the schedules), or used to optimise the parameters/settings of the main dynamic solver (e.g. in the supply-chain problem (Akanle

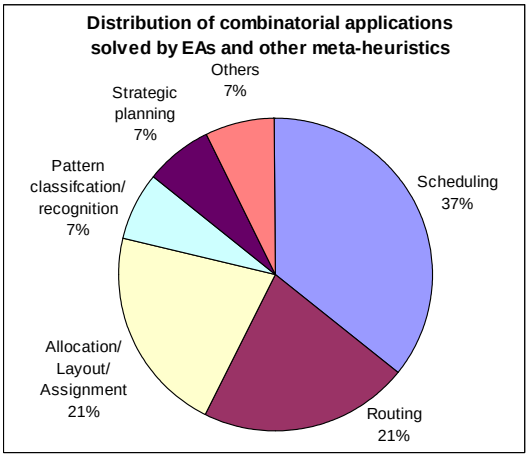

Figure 3.3: Distribution of combinatorial applications that are solved by EAs/metaheuristics.

& Zhang 2008), GA is used to optimise the parameters of the models that are used to coordinate the distributed decision-making process).

- EAs/meta-heuristics are used in an *offline way* (3% in the combinatorial domain and 4% in the continuous domain) to find some optimal policies/rules, which then will be used during the online optimisation process. One example is the problem of dimensioning and load balancing for multi-topology Interior Gateway Protocol traffic (Wang, Ho & Pavlou 2008). In this problem, GA is used to find the optimal link weights before the online optimisation process starts. These optimal link weights then will be used as the basis for the main solver to adjust the network topologies online as time goes by to react to environmental changes. It should be noted that in this group we only consider applications where online optimisation does exist, i.e. beside the offline policies/rules found by EAs/meta-heuristics, the solver still needs to optimise its solution to react to changes (like the example of (Wang, Ho & Pavlou 2008) above). Applications where the offline rules found by EAs are used online without any further optimisation, e.g. in (Huang & Wu 2008, Xiong 2008), will not be considered.

3.3 Coverage and Gaps in current EDO academic research

In this section I will discuss some findings, which show that current academic research has not yet covered all common types of DOPs and that there are many real-world DOPs where the current assumptions do not hold. The section will also identify the type of problem and characteristics

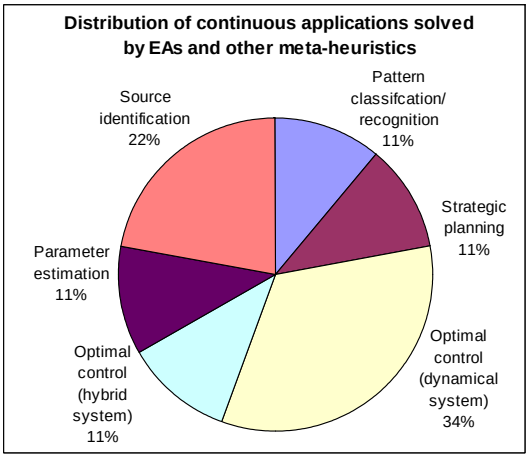

Figure 3.4: Distribution of continuous applications that are solved by EAs/metaheuristics.

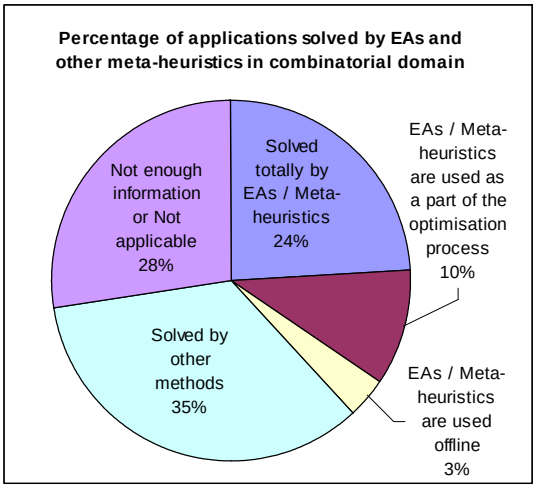

Figure 3.5: Percentage of applications solved by EAs and other meta-heuristics in the combinatorial domain

that have been covered by existing EDO academic research.

3.3.1 Continuous constrained problems

The first important class of problems that I found very common in the survey, but is not covered by most current EDO academic research, is the class of continuous constrained problems.

As can be seen in Figures 3.7 and 3.8, a majority of the surveyed applications are constrained problems (73% in the combinatorial domain and 74% in the continuous domain). The survey results in these two figures also show that a large number of constrained applications have dynamic constraints.

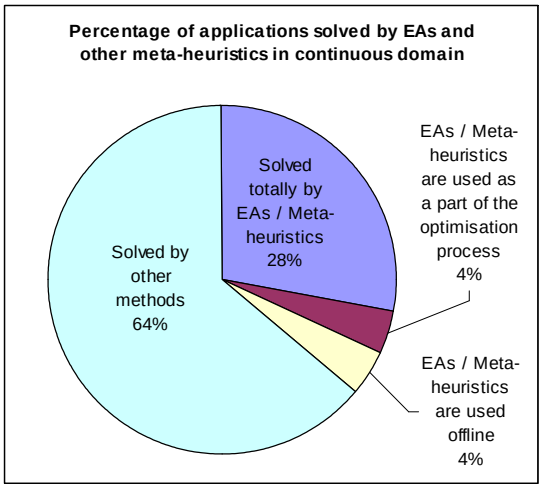

Figure 3.6: Percentage of applications solved by EAs and other meta-heuristics in the continuous domain

The type of continuous dynamic constrained problems, however, has almost not been studied in EDO continuous benchmark problems, as already reviewed in Section 2.3. Although almost all existing general-purpose EDO academic benchmark problems are unconstrained and domain constraint problems according to the literature review in Section 2.3, the survey results in Figures 3.7 and 3.8 show that, this type of problem occupies only 15% of the continuous surveyed applications and 0% of the combinatorial applications.

This lack of attention of current EDO research on continuous constrained problems can be considered an important research gap, which might question the usefulness of existing continuous dynamic optimisation algorithms in solving dynamic constrained problems, as they have been designed and tested in the unconstrained/domain constraint cases only. This issue will be further discussed at the end of this chapter and will be investigated further later in this thesis.

3.3.2 Time-linkage problems

Another important class of problems, which is very common in the surveyed real-world applications but has not received enough attention from current EDO academic research, is the class of time-linkage problems. As already mentioned in Subsection 2.1, a DOP is a time-linkage problem if its future dynamics depend on the decision made earlier by the solvers. In other words, a time-linkage problem is an online control problem where the algorithm is the actual controller to control the future behaviour of the system. Our survey results (Figures 3.9 and 3.10) show that a large number of the problems in the surveyed references were mentioned by

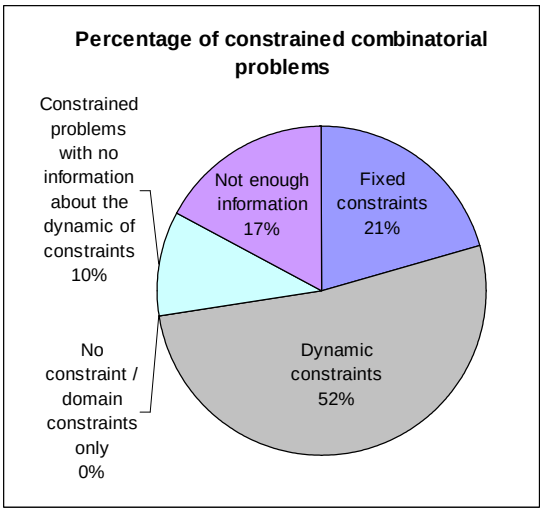

Figure 3.7: Percentage of constrained combinatorial problems

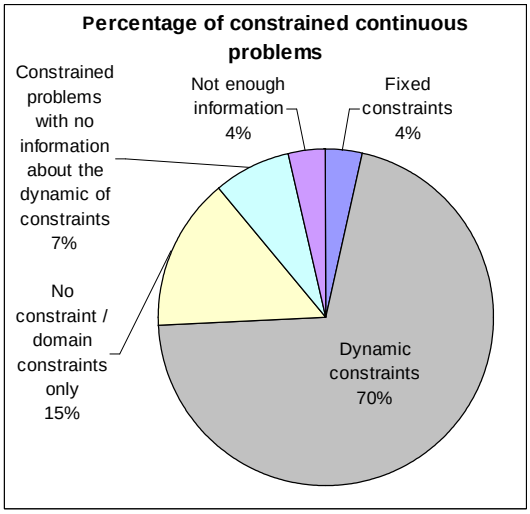

Figure 3.8: Percentage of constrained continuous problems

the authors as having the time-linkage properties (45% in the combinatorial domain, and 81% in the continuous domain).

Despite the popularity of time-linkage problems (as illustrated by the large number of applications found in our surveys - Figures 3.9 and 3.10), as mentioned in Subsections 2.1 and 2.3 this type of problem still has not attracted much attention from the EDO academic research community. Only very few recent studies proposed using EA to solve time-linkage problems (Branke & Mattfeld 2005, Bosman 2005, Bosman & Poutré 2007, Nguyen & Yao 2009*b*). There are also only few test problems with the time-linkage property in EA research (Ursem *et al.* 2002, Branke

& Mattfeld 2005, Bosman 2005, Bosman & Poutré 2007, Nguyen & Yao 2009*b*). This lack of attention creates an important gap, which should be addressed if we want to apply EAs effectively to solving real-world DOPs.

From the practical side, the time-linkage property has also not been studied sufficiently in evolutionary research. In the survey we found an interesting fact that although some references using EAs/meta-heuristics do mention the time-linkage property when solving real-world problems online (Kanoh 2007, Atkin *et al.* 2008, Moser & Hendtlass 2007*b*, Ngo *et al.* 2006, Wang, Ho & Pavlou 2008, Akanle & Zhang 2008, Jin *et al.* 2007, Rocha *et al.* 2005, Morimoto *et al.* 2007, Wang, Tao & Cho 2008, Sonntag *et al.* 2008), none of them equip their EAs with the ability to handle the time-linkage property online. In these references, the time-linkage property is either ignored, e.g. in (Jin *et al.* 2007, Kanoh 2007, Ngo *et al.* 2006, Moser & Hendtlass 2007*b*), or handled using a separate component/heuristics, e.g. in (Atkin *et al.* 2008, Akanle & Zhang 2008), or handled by EAs in an offline way, e.g. in (Wang, Ho & Pavlou 2008, Sonntag *et al.* 2008). The fact that all EA approaches from our surveyed real-world references do not handle the time-linkage property online demonstrates that applying EAs to solving real-world DOPs is still a challenge for the community.

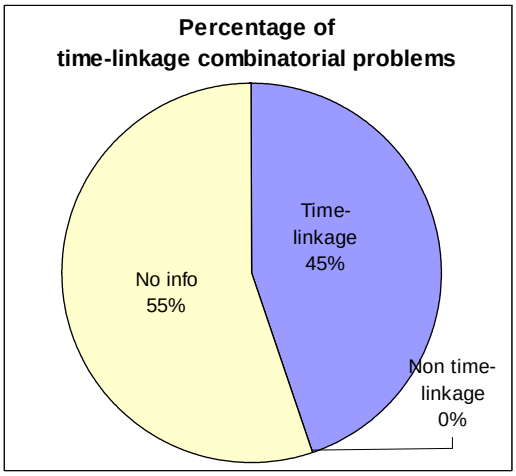

Figure 3.9: Percentage of time-linkage combinatorial problems

3.3.3 Optimisation goals

One of the possible gaps that I found between the current EDO research and the surveyed real-world DOPs is that some optimisation goals found in real-world problems might not entirely be

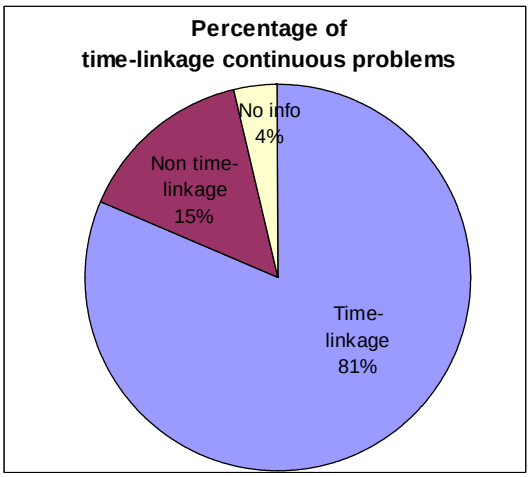

Figure 3.10: Percentage of time-linkage continuous problems

covered by academic EDO research.

Our survey shows that real-world DOPs can have many different optimisation goals:

- *Optimality*: Find the solutions with the best objective values

- *Quick recovery*: Find a decent solution as quick as possible or before a certain deadline. Because real-world DOPs are solved in real-time, this is an important goal for many different types of applications where there is a restriction in the amount of time that the solvers can use to produce a solution. It should be noted that although time restrictions have already been deployed in most current academic benchmark problems in the form of change frequency, this is not always the same as the deadlines to produce a good solution required in many real-world applications because such deadlines may occur *before* the next change. For example, for the airline schedule recover problem (Liu *et al.* 2007), the faster the algorithm to provide a new schedule, the more preferred it is regardless of the frequency of change; for the supply-chain configuration problem (Akanle & Zhang 2008), customer orders need to be complete before a given deadline regardless of when will the next change occurs. Many other examples can be found in Tables 1, 2, 3, 4 and 5 in the Appendix (page 246). To capture this optimisation goal benchmark problems might need a deadline parameter in addition to the change-frequency parameter.

- *Specification/requirement satisfaction*: Find the solution $s\left(t_{now}\right)$ so that future solutions $s\left(t_{now}+i\right) \subseteq S\left(t_{now}+i\right)\ \forall i = 1 : N$. This is a broad class of optimisation goals, of

which some can be encapsulated in the constraints of each static instance of the DOPs under certain circumstances. However, as will be discussed later it is not always possible to incorporate this type of optimisation goals as constraints of DOPs' static instances. Common examples of specification/requirement satisfaction goals are:

- *Previous-solution displacement restriction* : Find a new solution that is not much different from the old one from the previous time step. For example, in the aircraft taking-off/landing scheduling problems (Atkin *et al.* 2008, Bianco *et al.* 2006, Moser & Hendtlass 2007*b*, Wilkins *et al.* 2008), whenever an environmental change occurs and it is necessary to re-schedule the orders of aircrafts, the new schedule needs to be as close to the previous one as possible to minimise disruptions to the operations of other aircrafts. This goal is particularly common in combinatorial problems in the field of scheduling and routing. This is usually a secondary goal and is used alongside with the optimality goal.
- *Reference-solution displacement restriction* : as time goes by, find dynamic solutions so that the actual trajectory of solutions is not much different from a given reference trajectory. This goal is particularly common in control problems where a reference trajectory of solutions has been provided offline based on simulation, and the task of the online controller is to provide control decisions in real-time so that the system can follow the reference trajectory as closely as possible. For example, in the problem of controlling an electric ship power system (Mitra & Venayagamoorthy 2008), the solver needs to keep finding optimal solutions (control decisions) as time goes by to regulate the system's bus voltage to a pre-defined voltage value.
- *Reaching a specific target*: Online solutions need to be provided at each time step so that eventually the system will reach a specific target. This goal is common in path-finding problems where the task is to accumulatively create a route in real-time so that the route can finally reach a given destination (e.g. the robot planning problem (Mills-Tettey *et al.* 2008) or the real-time heuristic-search problem in AI games (Bulitko *et al.* 2007)). The goal is also common in dynamical systems (e.g. many chemical processes listed in Table 5 (page 263)) where a specified state is given as the target.

  - *Ensuring that future solutions are within certain bounds*: Online solutions need to be calculated at each time step in a special way so that they will NOT lead to future situations where all possible solutions are beyond certain bounds. For example, in the problem of controlling the vehicle speed of a Silverstone F1 racing car (Velenis & Tsiotras 2008), the solver needs to choose the appropriate solutions (control variables) so that (i) the F1 velocity will not exceed a "critical value" at any point in time, (ii) at the end of the current planning horizon the vehicle is guaranteed to come to a complete stop, and (iii) in case an obstacle exists after the current planning horizon, the vehicle can follow an "escape trajectory" to avoid collision. Other examples can be found in many control problems where the stability of the systems need to be maintained and in scheduling problems where the scheduler needs to make sure that the current schedule will not lead to future schedules where some of the tasks are infeasible to implement (see Tables 1, 2, 3, 4 and 5, page 246, for details). Usually this goal is used as a secondary goal alongside with another goal.

- *Combination of different goals.* In many real-world DOPs there might be more than one optimisation goal. Our survey results (Figure 3.11) show that at least 65% of the surveyed applications have more than one goal.

Figure 3.12 shows the high variety of different optimisation goals and the distribution of the goals in the surveyed applications. As can be seen in the figure, optimisation goals such as *optimality*, *quick recovery*, *displacement restriction*, *specification satisfaction* etc. are all important and they exist in a significant number of surveyed applications (Details of problems with the above optimisation goals can be found in the column "Optimisation goal" of Tables 1, 2, 3, 4 and 5 , page 246).

Despite this high variety of optimisation goals in real-world DOPs, it is unclear if current EDO academic research has covered all these goals. It seems that currently the main focus of EDO research is only on finding the optimum solutions/ optimum trajectory, which might not entirely reflect the aforementioned goals except the goal *optimality* and to some extent the goal *quick recovery*. For example, as already discussed in Section 2.2 most existing performance measures only evaluate the performance of the algorithm based on the fitness values of the solutions without considering how different the new solution is from the previous one; if there is

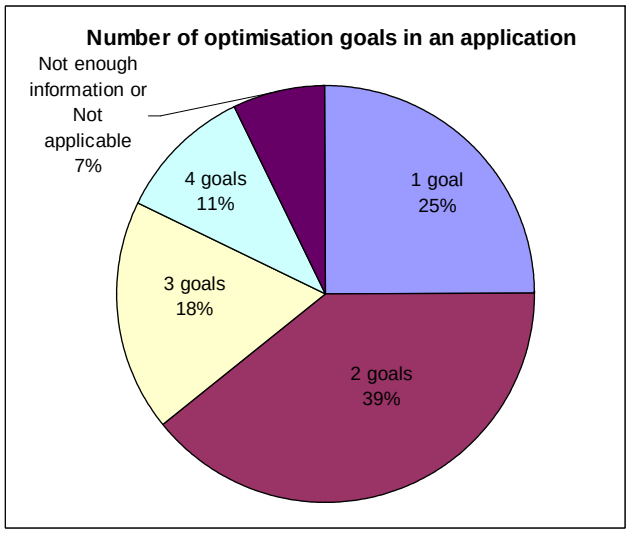

Figure 3.11: Distribution of applications with multiple optimisation goals

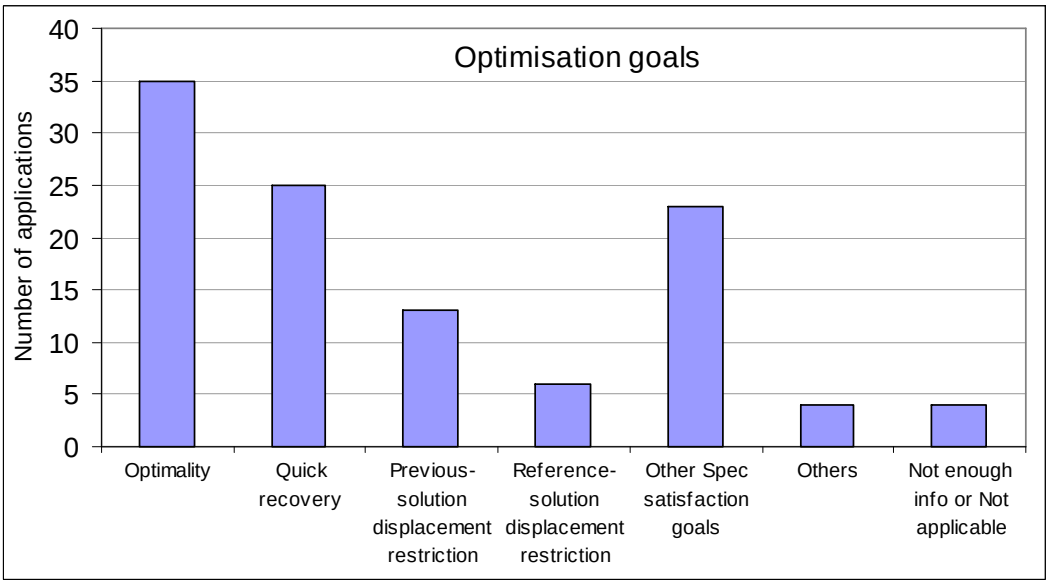

Figure 3.12: Distribution of different optimisation goals in the surveyed applications

any restriction/requirement for the future objective values; or whether the algorithm has met the deadline and how good is the performance before the deadline. Most current EDO benchmark problems, as reviewed in Section 2.3, also do not have any specification/requirements to support the goal *Specification/requirement satisfaction* and do not have any specific deadline except the change frequency to support the goal *quick recovery*.

One might argue that the goal *quick recovery* can be covered by some of the existing EDO

performance measures, for example the reactivity measures (Weicker 2003), which is aimed at evaluating the ability of algorithms to react quickly to changes and hence to some extent relates to the *quick recovery* optimisation goal that we have discussed above. However, even this measure might not be able to fully evaluate the performance of algorithms in satisfying the goal of quick adaptation in real-world problems. Firstly, this is because the measure is based on the assumption that there must be a drop in performance after a change, which might not be true in real-world situations. Secondly, the measure only evaluates how long it takes for an algorithm to recover to a certain level from a performance drop after change, while in many real-world situations that we surveyed it is required to evaluate how well the algorithms can adapt before a certain deadline. For this type of optimisation goal, the commonly used modified offline error/performance measure (Branke 2001*b*, Branke & Schmeck 2003) or the best-of-generation (Yang & Yao 2003) measure might still be useful, but it might be necessary to modify the measure by adding some deadline parameters, which are not necessarily the same as the frequency of changes.

This lack of focus on other optimisation goals in EDO research might create a gap between academic research and real-world applications because it would be difficult to evaluate whether existing academic DO methods can satisfy other optimisation goals beside the goal *optimality*. Of course in certain cases it might be possible to integrate the other optimisation goals into the objective/fitness function or as some type of constraints, and hence the problem can be solved with a single goal: *optimality*. However, such situations have also not been captured in existing benchmark problems and performance measures as shown in the review in Chapter 2. In addition, it might not always be possible to integrate multiple goals into the objective/fitness function, as will be discussed below.

The gaps between academic research and real-world applications in optimisation goals exist not only in the way academic performance measures/benchmark problems are proposed, but also in the way EAs are designed. Currently most dynamic optimisation EAs are designed to work well with the current measures/benchmark problems, i.e. to find the best objective values. Such design approaches, however, might not work well to entirely satisfy the other optimisation goals in real-world problems. For example, the best solution an algorithm achieves after a change to meet the optimality goal might be totally different from the solution found before change. In this case the displacement restriction goal would not be achieved.

The need to fill the gaps in dealing with such optimisation goals as displacement restriction and specification satisfaction is even more important in solving time-linkage problems. The detailed survey results in Tables 1, 2, 3, 4 and 5 show that these optimisation goals are very common in time-linkage problems. It is also much more challenging to satisfy these two goals in the time-linkage case. In the non-time-linkage case these two goals above can be integrated into the objective function as constraints and algorithms only need to take into account the constrained objective value at the current time step to find a satisfiable solution. In the time-linkage case, however, it is much harder because algorithms need to not only calculate the current objective value but also to predict the outcome of the system in the future given the current solutions.

The points above show us that the lack of appropriate tools to satisfy the aforementioned optimisation goals, at least as secondary optimisation goals or constraints beside *optimality*, might be a challenge for EA research to solve real-world DOPs. It also shows the necessity to pay more attention on designing new algorithms/performance measures to deal with these newly identified goals.

3.3.4 Factors that change

Other real-world DOP characteristics that have not been fully captured in existing EDO research are the factors that change. As shown in Subsection 2.3, most existing general-purpose academic benchmark in current EDO research only represent changes in the objective function. However, our survey results (Figure 3.13) show that in real-world DOPs there are also other factors that change. They are changes in constraint functions, changes in number of variables, changes in domain range, and switch-mode changes. Of these, constraint changes are the most common, followed by changes in number of variables and switch-mode (the system switches from one mode/dynamic model to another) changes. Changes in objective functions and constraints are common in both the continuous and combinatorial references and occur in a wide range of applications. Changes in number of variables (dimensional changes) are more common in the combinatorial problems (account for two-third of the total number of applications with dimensional changes) and also occur in a wide range of applications. In the continuous domain, changes in number of variables occur mainly in the hybrid systems, which are also the applications where all of the switch-mode and domain-range changes take place.

The fact that beside objective-function changes, other type of changes are also common suggest that more research should be focused on designing benchmark problems with changes in constraints, number of variables and domain ranges. Benchmarks simulating hybrid systems might also be necessary to solve this type of applications effectively.

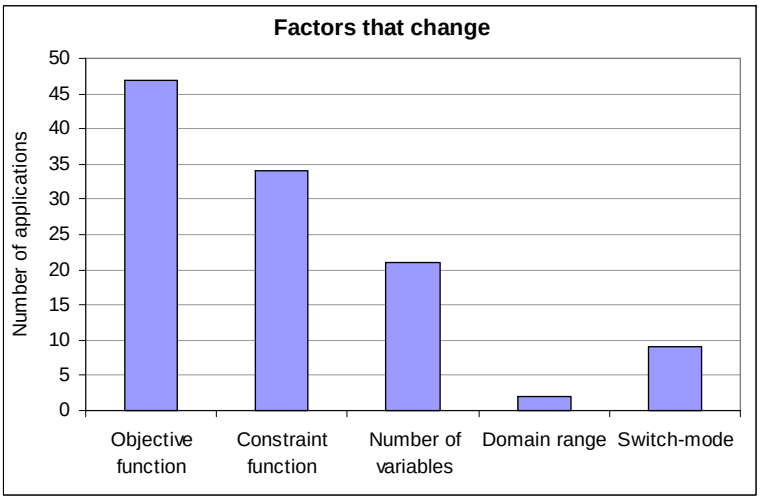

Figure 3.13: Distribution of typical changing factors in the surveyed applications

3.3.5 Problem information

In EDO academic research, beside the general assumption that there are some correlations between the environment before and after a change, different research have different assumptions about whether changes are visible; whether it is easy to detect changes; if changes are predictable and if the changes are recurrent. However, as we have discussed in Chapter 2 there is not much evidence of whether these assumptions are true in real-world scenarios, and if these assumptions are true, how common they are in real-world applications. In this section I will investigate the existence and popularity of these assumptions in the set of real-world applications we selected for the survey.

Visibility and detectability of changes

The first two characteristics to be investigated are the visibility and detectability of changes from the perspective of the optimisation algorithms. The visibility and detectability of changes are important factors which determine the performance of an algorithm if this algorithm is *reactive*, i.e. if it relies on change detection to respond to changes. For example, algorithms following the *diversity-introducing* and *memory-introducing* strategies as reviewed in Subsection

2.1 will not be able to do well if a change is not visible or detectable. It should be noted that the terms "visibility of changes" and "detectability of changes" that we use here are viewed from the perspective of the optimisation algorithm only. They show whether the algorithm is informed about a change in the fitness landscape and whether it is easy to detect changes in the fitness landscape. These visibility and detectability of changes in fitness landscapes *might*, or *might not*, relate to the actual visibility and detectability of real-world environmental changes from the perspective of the real-world system. For example, from the perspective of the solar powered rover robot (the real-world system) in (Mills-Tettey *et al.* 2008), changes in the real-world dynamic environment (unexpected obstacles in the pre-planned path) are not known (*invisible*) and hence the robot *needs to detect* those changes using its sensors. However, from the perspective of the optimisation algorithm (the built-in navigator software), which is a part of the robot system, corresponding changes in the fitness landscape are *known* and hence there is *no need to detect*, because information about external environmental changes were transferred directly from the sensors to the fitness landscape of software.

Regarding to these two characteristics, currently in EDO research the main assumption adopted by many existing EDO academic research is that changes either are known (visible) or are easy to be detected by using *just one or a few or a fixed set* of detectors. As a result, many current *reactive* dynamic optimisation algorithms (algorithms that only respond when they knows a change occurs) are designed without a change detection scheme or with a simple change detection scheme which monitors the changes in values of average best-performance e.g. (Cobb 1990, Vavak *et al.* 1997*a*), or some members of the population e.g. (Branke 1999, Hu & Eberhart 2002) or members of the memory set e.g. (Yang 2005*a*, Simões & Costa 2007), or a fixed point e.g. (Carlisle & Dozier 2000) or of a random solution e.g. (Singh *et al.* 2009). Only very few studies (Morrison 2004, Richter 2009) take into account the situations where it might not be possible to detect changes using a few detectors because changes might occur only in a part of the search space.

Despite its popularity, there is no clear explanation of why the assumption that changes can be easily detected is accepted in most EDO academic research and whether the assumption is true in real-world situations. One of the possible reasons might be that this assumption is true in almost all existing artificial benchmark problems, except a few benchmark sets proposed recently (Nguyen & Yao 2009*a*, Richter 2009, Richter 2010).

To verify if the considered assumption is really true in real-world situations, at least in the surveyed set of applications, in this subsection I will analyse the visibility and detectability of changes in the fitness landscapes of the surveyed applications. It is hoped that the analysis will give a better justification on the use of the assumption above. It is also hoped that the analysis will provide us with a better understanding of how changes are handled in real-world applications and whether there is any type of changes not covered by the current assumption. Results of my analysis on the visibility of changes in the surveyed applications are shown in Figure 3.14, and results of my analysis on the detectability of changes in these applications are shown in Figure 3.15. Details of the visibility and detectability of each of the surveyed applications can be found in Tables 1, 2, 3, 4 and 5 in the Appendix.

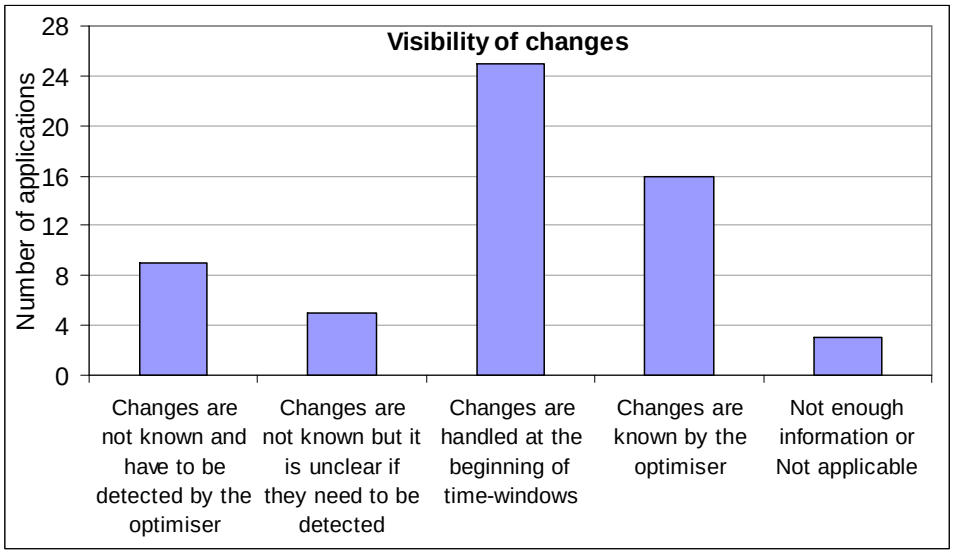

Figure 3.14: The visibility of changes (from the perspective of optimisation algorithms) in the surveyed applications

Figure 3.14 shows that for a majority of the surveyed applications, it is indeed not necessary to detect changes constantly. In many cases (16/56 applications) changes are known by the optimisation algorithms. In many other cases (25/56 applications), the optimisation process are divided into time-windows where during each time-window the environment is considered static. Due to that, in time-window approaches the dynamic problems are transformed into a sequence of static problems, each starts at the beginning of a time-window and hence changes detection are also not needed.

However, in the cases where it is necessary for the optimisation algorithm to detect changes,

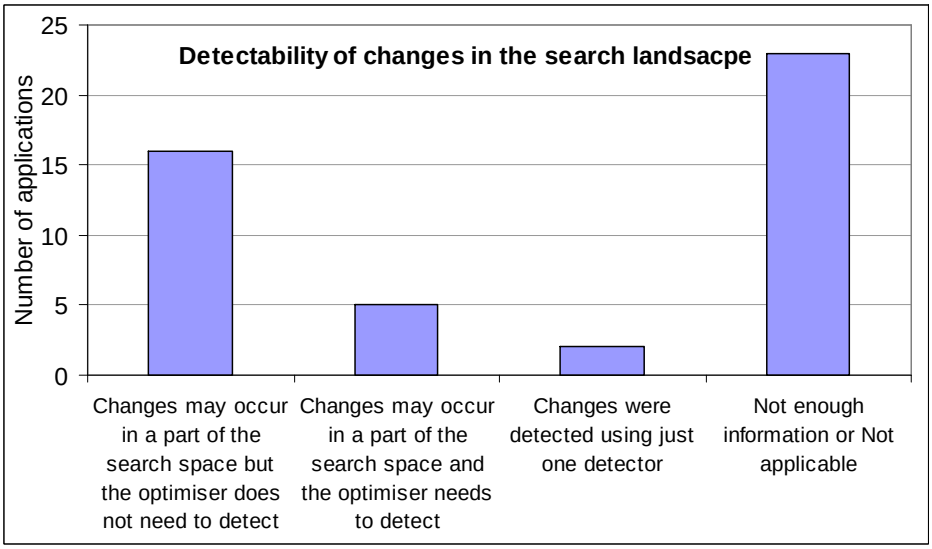

Figure 3.15: The detectability of changes in the search landscape (from the perspective of optimisation algorithms) in the surveyed applications

Figure 3.15 shows that it is not always possible to detect changes using just one or a few detectors. Among nine applications where optimisation algorithms need to detect changes, only two use one single detector for change detection. The data from Figure 3.15 suggests that the reason for many applications not to use only one single detector might be that in many applications changes might occur in only a part of the search space. As can be seen in Figure 3.15 there are 21 applications where changes might occur in a part of the search space, of which five have changes that the optimisation algorithms need to detect.

The survey results suggest that although in a majority of cases a simple change detection method is sufficient, for certain applications it might be necessary to have a sophisticated change detection method to detect changes effectively. Because complicated change detection method might also add unnecessary cost to the algorithm when it is used to solve problems where changes are known (there are many problems like this in the survey), in designing future algorithms it would be better to separate the change detection mechanism from the search mechanism so that in case changes are visible the algorithms do not need to spend any unnecessary effort on detecting changes.

Predictability of changes

The third characteristic to be investigated in this subsection is the predictability of changes. This characteristics has not been studied well in EDO academic research. As can be seen in the review in Section 2.3, many current academic research with artificial benchmarks either assume that changes are not predictable or do not take into account the predictability of changes in their default settings. There are only a few studies (Bosman 2005, Hatzakis & Wallace 2006, Rossi *et al.* 2008, Zhou *et al.* 2007, Simões & Costa 2009) on the predictability of changes in EDO academic problems. However, the survey (Figure 3.16) shows that a large number of applications (61%) have at least a part of their changes predictable. This large proportion of predictable changes suggest that the predictability property should be taken into account when designing dynamic optimisation algorithms and benchmark problems. Because many of the predictable changes in the surveyed applications are problem-specific, it would also be useful to design predictable changes that are tailored to specific applications.

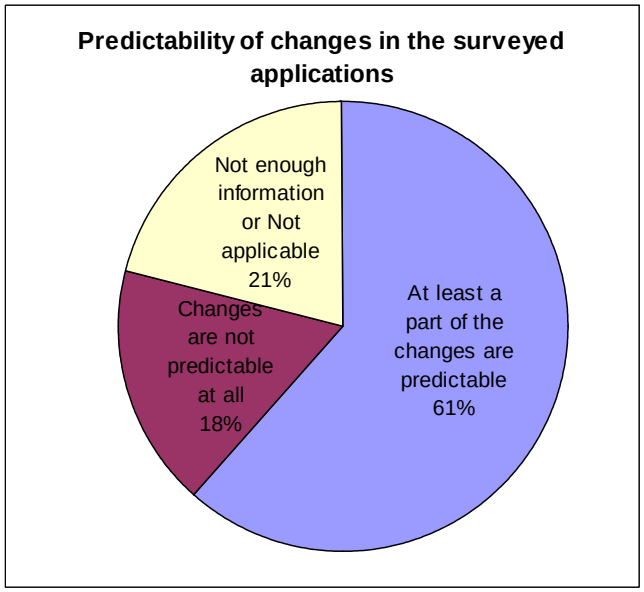

Figure 3.16: The predictability of changes in the surveyed applications

Recurrence of changes

Another important characteristic, which is believed to be common in DOPs, is the recurrence of changes. The assumption that changes might some how re-occur in dynamic environments is the key point leading to the development of algorithms following the *memory-based approach*,

one of the major approaches in EDO, and the development of some prediction methods, e.g. (Simões & Costa 2009) to anticipate the recurrence of changes. Because of that, it is important to investigate how common this characteristic is in real-world applications. In the set of surveyed application, the survey results (Figure 3.17) show that 25% of the applications have some kind of recurrent/periodical/cyclic changes. It means that the use of memory might be suitable for these types of problems and confirms the usefulness of memory-based approaches in solving certain types of real-world applications.

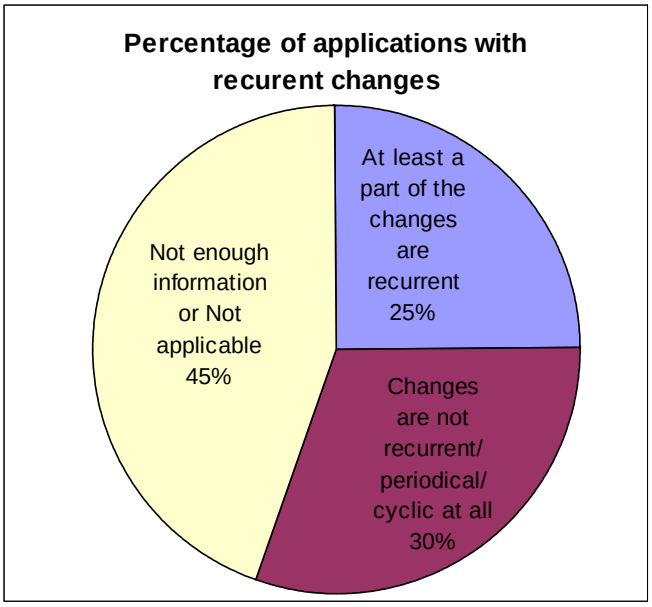

Figure 3.17: Percentage of applicatins with recurrent changes

3.3.6 The way problems are solved

When to solve a problem online

The survey provides interesting observations about the two types of situations where real-world dynamic problems are solved online. First, problems are solved online when it is not possible to provide a complete model to represent the dynamic of the system due to the lack of knowledge about the system's future behaviours (e.g. if there is any change in parameter values compared to the designed values, of if there is any unexpected event etc.). Instead it is only possible to model the current state of the system given the available knowledge and to provide an optimal solution for this current state. In such case, it is necessary to solve the problem online as time goes by, so that whenever the system changes its state, we can provide a new optimal solution for

the newly changed state. This is the most common case that we found in real-world applications. This situation correlates to the definition of Bosman (2007), which states that a DOP is said to be online "if the dynamic optimization function cannot be evaluated for all future times $t > t^{now}$ and the dynamic optimization problem must therefore be solved as time goes by".

Second, problems are also solved online when there is a complete model to represent the dynamic of the real-world system *but* it is either computationally expensive to solve the problem offline given the complete model, or there is some rules/regulations that prevent the solver from solving the problem offline. In such cases, the solver needs to solve the problem online, either to save the computational cost, or to satisfy some regulations although theoretically given enough time the problem can be solved offline. This situation is very common in virtual reality applications where although complete information about the virtual environment are available, the problems are still needed to be solved online for some reasons: first, revealing complete environment information to optimisers (virtual agents in this case) is considered cheating in virtual reality applications like commercial games (Bulitko *et al.* 2007); second, allowing virtual agents to know the complete environment might lead them to do actions beyond the designed ability of the roles they play (Dini *et al.* 2006); third, the optimisation problem with complete information may become too complex and hence too computationally expensive to satisfy the time constraint in real-time games (Orkin 2006). Another interesting example of problems being solved online even when it is possible for them to be solved offline is the problem of planning future farming strategies (Jin *et al.* 2007). In this case, although the complete model to represent future dynamics can be theoretically created, the problem is solved online to study how the dynamic environment affects the outcomes of different solutions.

When to re-use previous knowledge and when to restart

It has been shown that the knowledge re-using (tracking) approach, or specifically "somehow use knowledge about the previous search space to advance the search after a change" (Jin & Branke 2005), can be used to speed up optimisation in dynamic problems. The survey confirms that the knowledge re-using approach is indeed very common in real-world DOPs (72% as shown in Figure 3.18).

The survey also shows an interesting observation about the reason for knowledge re-using to be chosen over restarting. Although in most academic evolutionary research on artificial

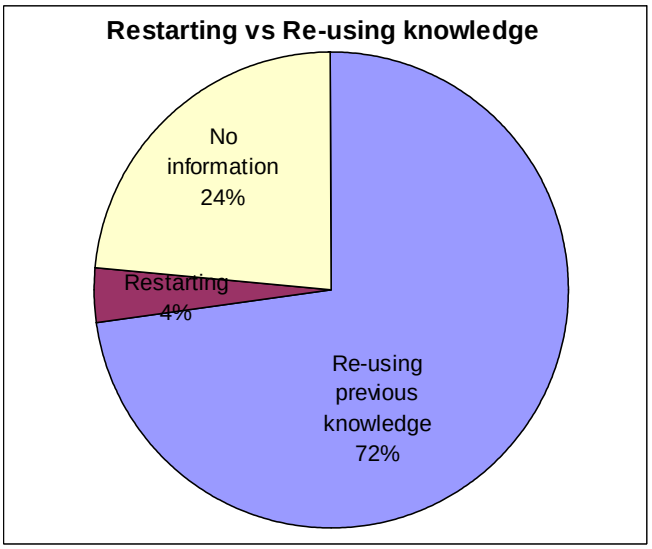

Figure 3.18: Percentage of applications that adopt the tracking approach, compared to those adopting the restarting approach.

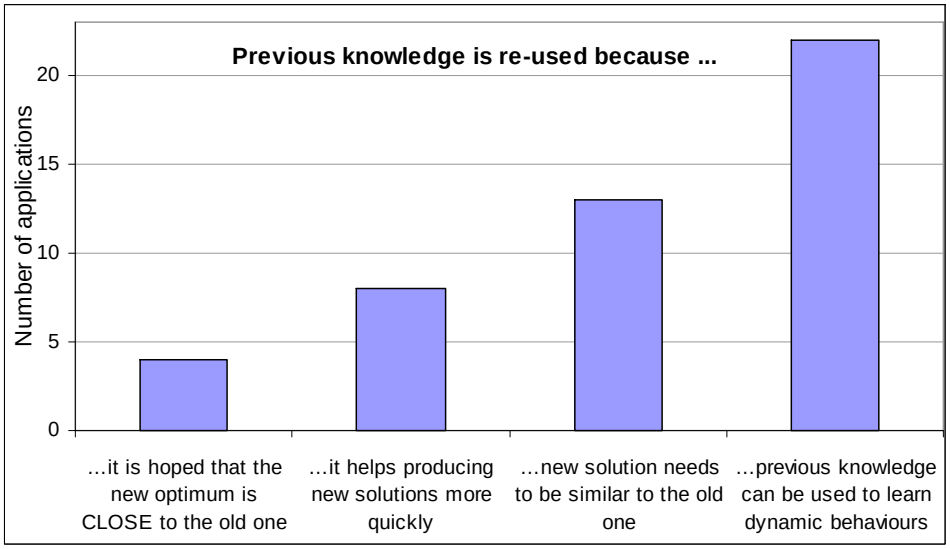

Figure 3.19: The reasons for the surveyed applications to use the tracking approach.

benchmark problems, knowledge re-using is chosen over restarting because the global optimum after a change is closed to the global optimum or another local optimum before the change (which is a property of most academic benchmark problems), the survey shows that this is not the only reason for choosing the knowledge re-using approach. In fact, as can be seen in Figure 3.19, among all applications that follow the knowledge re-using approach, only 8.33% clearly state that knowledge-reusing is chosen because it is hoped that the new global optimum is close

to the old one. Other reasons for real-world practitioners to choose the knowledge re-using approach over the restart approach (even when the restart approach may produce better quality solutions as mentioned in (Araujo & Merelo 2007, Mertens *et al.* 2006)) are as follows (note that one application can have more than one reasons):

1. Knowledge re-using is chosen because it helps producing new solutions more quickly: 16.67%
2. Knowledge re-using is chosen because there is a displacement restriction (new solution needs to be similar to the old one): 27.08%
3. Knowledge re-using is chosen because previous knowledge can be used to learn dynamic behaviours: 45.83%

Three reasons above correlate to my previous findings of the uncaptured optimisation goals mentioned in Subsection 3.3.3. The popularity of these optimisation goals and the corresponding reasons for re-using knowledge suggest that more attention might be needed in academic EDO research to address this issue. Particularly, more benchmark problems and performance measures should be developed to reflect the new optimisation goals and to evaluate the performance of optimisation methods that re-use previous knowledge for other reasons than the possible close distance between the old and the new global optimum.

Single-objective vs multi-objective

The final observation that I got from the survey is the more popularity of single-objective approaches compared to multiple-objective approaches. Although a large number of applications have more than one optimisation goals (as shown in Subsection 3.3.3), it is interesting to see that only 11% of the applications are solved as multi-objective problems, compared to 73% solved as single-objective problems. There might be different reasons for this, but as shown in (Atkin *et al.* 2008), one interesting reason for single-objective approaches to be preferred is that it simplifies users' tasks (they do not need to choose between multiple optimal solutions). This might give algorithm designers some ideas of whether and how multi-objective approaches should be used in practical applications. It should be noted that in our survey, we categorise as single-objective applications those applications that combine multiple objectives into a single one using aggregation methods e.g. (Morimoto *et al.* 2007, Atkin *et al.* 2008).

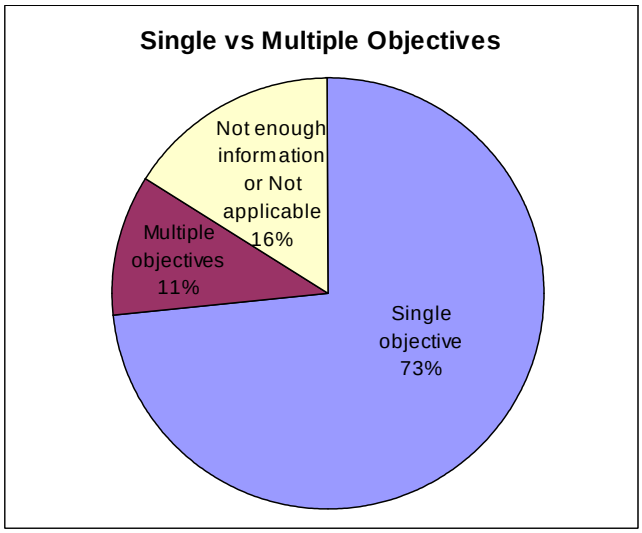

Figure 3.20: Percentage of applications that adopt the single-objective approach, compared to those adopting the multiple-objective approach.

3.4 Summary

The literature reviews in Chapter 2 have shown that current EDO academic research focuses on certain classes of DOPs, which are assumed to have some special characteristics. The literature reviews also showed that it however remains a question of whether these special characteristics fully represent real-world applications, and if they do not, what are the other real-world characteristics that have not been captured in EDO academic research.

This chapter contributes to answering the above question by comprehensively reviewing a large set of recent real-world dynamic optimisation references. Based on the results of this review, which has never been done before, I have investigated for the first time the links between EDO academic research and real-world DOPs. First, I recognised the areas of applications where the current EDO assumptions hold. Second, I pointed out certain gaps between academic EDO research and real-world DOPs. These gaps include (a) common classes of real-world problems that have not attracted much interest from the community and (b) common aspects in real-world dynamic optimisation that have not been captured in the current assumptions of academic EDO. Based on this review, I also discussed the necessity and possibility to extend current EDO research to better reflect the characteristics of real-world problems and to solve real-world DOPs more effectively.

The contributions of this chapter can be summarised into three groups as follows:

1. *Identifying the common types of real-world problems, the coverage (or the lack thereof) of current EDO academic research on each type of problem, and providing suggestions on how to close the gaps*: Current studies in academic EDO do not cover all types of common dynamic optimisation problems yet. The most common type of DOPs that current academic research considers are unconstrained, non-time-linkage problems. However, this type of problem occupies only a small part of the surveyed applications. I found that there are two other types of problems that are very common in real-world applications but received very little attention from the community:

   (a) *Continuous constrained problems*: A major number of surveyed problems are constrained problems. However constrained problems have not been considered in the majority of current continuous dynamic optimisation research.

   (b) *Time-linkage problems*: A large number of surveyed problems are time-linkage problems. However there is very little research on this type of problem in EDO academic research.

2. *Identifying the representative characteristics of real-world DOPs, whether they have been captured by EDO academic research and providing suggestions on how to close the gaps*

   (a) *Optimisation goals*: Although many of current EDO academic research works only focus on one single optimisation goal: *optimality*, in the surveyed applications there are many other common optimisation goals as *quick recovery*, *previous-solution displacement restriction*, *reference-solution displacement restriction, reaching a specific target, ensuring that future solutions are in certain bounds*, etc. In order to solve real-world DOPs better, EDO research should take into account these optimisation goals, at least as secondary goals or special types of constraints when designing performance measures, benchmark problems and algorithms for solving certain types of DOPs.

   (b) *Factors that change*: Although most current EDO artificial benchmark problems have only one changing factor: the (parameters of) objective function, the survey shows that there are also other common types of changing factors: constraints, number of

variables, domain ranges and switch-mode changes. The analysis in this chapter also shows the distribution of each changing factor and the type of applications where the changing factors are common.

(c) *Problem information*: The chapter also investigates the types of real-world DOP's information that can be used by the optimisation algorithm to advance the search. These types of information include the *visibility*, *detectability*, *predictability* and *recurrence* of changes. For each type of information we also analysed their properties, their popularity, and whether the type of information has been studied in academic research.

3. *Identifying the ways real-world DOPs are solved, whether they are the same as in academic research, and providing suggestions on how to close the gaps.* The review and analysis was focused on three aspects

   (a) When to solve the problem online,

   (b) When to re-use previous knowledge, and why,

   (c) The proportion of single-objective approaches compared to multi-objective approaches

Summarising, the review in this chapter shows that besides the characteristics and assumptions commonly used in EDO academic research, real-world DOPs also have other important types of problems and problem characteristics that have not been studied extensively by the EDO community. In order to solve real-world DOPs more effectively, it is necessary to take these characteristics and problem types into account when designing new algorithms, performance measures and benchmark problems.

3.5 The gaps to be studied in this thesis

The review and analysis in this chapter shows that there are many open topics and gaps in EDO where further studies can be made to advance knowledge in the field. Within the scope of this thesis, I would like to focus my study on two main topics: *dynamic constrained optimisation* and *time-linkage optimisation*. As shown in the review of real-world applications, dynamic constrained optimisation problems (DCOPs) and dynamic time-linkage problems (DTPs) are the two most common types of problems among the surveyed applications, but they have not

attracted much attention from the community. In order to make these important classes of problems more accessible to the community, it is necessary to study the characteristics of the problems, the suitability of EAs in solving them and developing new EA techniques. It is hoped that my investigations on the characteristics of these two types of problems and my development of new algorithms to solve these two types of problems will contribute to closing the related gaps between academic research and real-world applications. The outcomes of the investigations are new benchmark problems and measures to reflect the uncaptured characteristics, and new algorithms to solve DCOPs and DTPs more effectively.

In addition, the reviews in this chapter and in Chapter 2 show that existing definitions of DOPs might not be sufficient to fully represent the characteristics of real-world DOPs. First, with the identifications of the not-well-studied types of problems as DCOPs and DTPs, and with the identifications of other changing factors as constraints, domain ranges, number of variables, it becomes clear that existing definitions (briefly reviewed in Chapter 1) might not be detailed enough to cover the newly identified problems and problem characteristics. Second, as analysed in Chapter 1, most formal definitions of DOPs do not clearly distinguish between a time-dependent problem and a DOP, which is a time-dependent problem that is solved in a dynamic way. Third, existing definitions do not take into account the fact that in many DOPs, the dynamic behaviours of the problem is influenced or decided by the optimisation algorithm, and hence defining/describing a problem without considering the solver might not sufficiently reflect the dynamics of the problem. These three reasons motivate me to develop a new definition framework to better represent DOPs. This definition framework also forms a part of this thesis.

Another related research that I have made during my PhD study to close the gap between real-world DOPs and academic research is to develop new algorithms specifically for the situations where changes cannot be detected easily (this situation has been discussed in Subsection 3.3.5). A new algorithm with promising initial results (equal to the best results from state-of-the-art methods) has been developed and described in (Nguyen 2008*b*, pp. 4-12).

Chapter 4

A definition framework for DOPs

4.1 Research gaps and motivations

Although the importance of DOPs has been shown through their presence in a broad range of real-world applications, due to the lack of research attention there are still many aspects that we do not fully know about this class of problems. One of the remaining questions is how should we define DOPs in detail to (i) distinguish DOPs from other types of time-dependent problems; (ii) encapsulate the behaviours of the dynamics and the types of dynamics (e.g. time-linkage vs non-time-linkage) in DOPs; (iii) encapsulate the changing factors (e.g. changes in objective function, constraints, domain range, dimension); and (iv) separate the static factors from the dynamic factors.

In Chapter 1, we have briefly discussed existing formal definitions for DOPs and categorised them into two types. The first type of definition defines a DOP as a sequence of static problems linked up by some dynamic rules (Weicker 2000, Weicker 2003, Aragon & Esquivel 2004, Rohlfshagen & Yao 2008, Rohlfshagen & Yao 2010). The second type of definitions defines a DOP as a problem that have time-dependent parameters in its mathematical expression (Bäck 1998, Bosman 2007, Woldesenbet & Yen 2009, Yu *et al.* 2010). A description of one such definition has also been given in Equation 1.1 (page 2, the definition of Bosman (2007)).

My reviews in the previous chapters, especially in Chapter 3, however have shown that the existing definitions of DOPs as cited above might not be detailed enough to fully represent the common characteristics of DOPs because of the following reasons:

1. *Existing formal definitions do not distinguish DOPs from other time-dependent problems*: As discussed in Chapter 1, although in some general (and hence less formal) definitions (Jin & Branke 2005, Morrison 2004, Branke 2001*b*) additional explanations are given to properly restrict the scope of DOPs from other time-dependent problems, most existing formal definitions of DOPs consider DOPs and time-dependent problems the same. In fact these formal definitions do represent time-dependent problems rather than DOPs. To avoid ambiguity and to define DOPs precisely, it is necessary to provide a formal definition which properly and clearly define DOPs as *time-dependent problems that are solved online in a dynamic way*.

2. *Existing formal definitions might not be detailed enough to represent different changing factors found in real-world applications*: Both groups of existing formal definitions that we mentioned above are not detailed enough to specify what are the changing factors in DOPs. In the group of definitions where DOPs are defined as sequences of static problems, there is no detailed specification of how a static problem can transform into another (as in (Weicker 2000, Weicker 2003, Aragon & Esquivel 2004, Rohlfshagen & Yao 2008, Rohlfshagen & Yao 2010), or the specifications are unrealistic (e.g. there is no evidence that the dynamic rules specified in (Weicker 2003) such as coordinate transformations, fitness rescales and coordinate stretchings are representative in real-world applications). In the group of definitions where DOPs are defined as problems that have time-dependent parameters in their mathematical expression (Bäck 1998, Bosman 2007, Woldesenbet & Yen 2009, Yu *et al.* 2010), the dynamics of the problem are just generally captured by including an additional parameter t (the time) in the static expression of the objective function (e.g. see Equation 1.1, page 2). Such a representation of dynamics might not be detailed enough to identify what factors are changed, how frequent the changes are, and what are the rules of changes.

3. *Existing formal definitions might not be detailed enough to encapsulate the time-linkage property of real-world applications*: Although the time-linkage property is very common in real-world problems (Subsection 3.3.2 shows that a majority of the surveyed problems have this property), most existing definitions of DOPs in academic evolutionary research do not consider this property. The only EC study where this property is described is the

research of Bosman (Bosman 2005, Bosman 2007). However, even in the DOP definitions in these references, the time-linkage feature is not explicitly expressed (see Equation 1.1, page 2). Instead, that feature is encapsulated in the expression of $f_{\gamma(t)}$. It would be better if the time-linkage property can be captured explicitly in the definition. In addition, none of the existing definitions encapsulates the important property of dynamic time-linkage problems (DTPs): *algorithm-dependent*. We consider DTPs algorithm-dependent because the structure of a DTP in the future may depend on the current value of $x(t)$, which in turn depends on the algorithm used to solve the problem. Because of this property, we believe that in order to define a DTP in an unambiguous way, the algorithm used to solve a problem instance should be considered a part of the problem instance itself.

To contribute in closing the gaps above, in this chapter I will propose a new definition framework which describes DOPs in a more detailed level. It is hope that the framework will help defining and characterising DOPs better and can be used as a basis for future theoretical works. The definition framework can also help generating benchmark problems that are able to capture the characteristics of DOPs, as I will describe in the next chapter. Within this chapter I will focus on the single-objective case only. Details of the definition framework will be described below.

4.2 A definition framework for DOPs

Definition 4.1 (Full-description form) *Given a finite set of functions* $F = \{f_1(x), ..., f_n(x)\}$*;* *a* full-description form *of* F *is a tuple*

$$\left\langle \widehat{f}_\gamma(x), \{\mathbf{c}_1, ..., \mathbf{c}_n\} \right\rangle$$

where $\widehat{f}_\gamma(x)$ *is a mathematical expression with its set of parameters* $\gamma \in \mathbb{R}^m$*, and* $\{\mathbf{c}_1, ..., \mathbf{c}_n\}, \mathbf{c}_i \in \mathbb{R}^m$ *is a set of vectors; so that:*

$$\widehat{f}_\gamma(x) \overset{\gamma=\mathbf{c}_1}{\rightarrow} f_1(x) \tag{4.2}$$

$$... \tag{4.3}$$

$$\widehat{f}_\gamma(x) \overset{\gamma=\mathbf{c}_n}{\rightarrow} f_n(x)$$

Each function $f_i(x), i = 1 : n \in N^+$ *is called an* instance of the full-description form *at* $\gamma = \mathbf{c}_i$. *From now on we will refer to the full-description form* $\left\langle \widehat{f}_\gamma(x), \{\mathbf{c}_1, ..., \mathbf{c}_n\} \right\rangle$ *as* $\widehat{f}$.

Example 4.4 *The combination of the expression* $\widehat{f} = ax + b$ *and the following set of parameter values for* a *and* b: $\{\{a = 1, b = 0\}, \{a = 0, b = 1\}, \{a = 1, b = 1\}\}$ *is the full-description form of the following set of functions:* $\{f_1 = x; f_2 = 1; f_3 = x + 1\}$.

The implication of a full-description form is that it can be used to represent different functions at different times by changing the parameters. It should be noted that, however, a full-description form is not unique: one set of functions can be represetend by multiple full-description forms and one full-description form can be used to represent mulitple set of functions. What is unique is a combination of (a) a full-description form $\widehat{f}$; (b) a given set of functions $\{f_1(x), ..., f_n(x)\}$ represented by $\widehat{f}$; and (c) the way the parameters of $\widehat{f}$ can be changed to tranform f_i to f_j $\forall i, j = 1 : n$. In real-world problems, changes in the parameters are usually controlled by some specific time-dependent rules or functions. For example, in dynamical systems changes of parameters can be represented by a linear, chaotic or other non-linear equations of the time variable t. The dynamic rules that govern how the parameters of a full-description form change can be defined mathematically as follows.

Definition 4.5 (Dynamic driver) *Given a tuple* $\left\langle \widehat{f}, \gamma_t, t \right\rangle$ *where* t *is a time variable,* $\widehat{f}$ *is a full-description form of the set of functions* $F = \{f_1(x), ..., f_n(x)\}$ *with respect to the set of* m*-element vectors* $\{\mathbf{c}_1, ..., \mathbf{c}_n\}, \mathbf{c}_i \in \mathbb{R}^m$, *and* $\gamma_t \in \mathbb{R}^m$ *is an* m*-element vector containing all* m *parameters of* $\widehat{f}$ *at the time* t;
we call a mapping $D(\gamma_t, t) : \mathbb{R}^m \times \mathbb{N}^+ \longrightarrow \mathbb{R}^m$ *a* dynamic driver *of* $\widehat{f}$ *if*

$$\gamma_{t+1} = D(\gamma_t, t) \quad \in \{\mathbf{c}_1, ..., \mathbf{c}_n\} \, \forall t \in \mathbb{N}^+ \tag{4.6}$$

and

γ_{t+1} *is used as the set of parameters of* $\widehat{f}$ *at the time* $t + 1$

Definition 4.7 (Time-dependent problem) *Given a tuple* $\left\langle \widehat{f}, D(\gamma_t, t) \right\rangle$ *where* t *is a time variable,* $\widehat{f}$ *is a full-description form of the set of functions* $F = \{f_1(x), ..., f_n(x)\}$ *with respect to the set of* m*-element vectors* $\{\mathbf{c}_1, ..., \mathbf{c}_n\}, \mathbf{c}_i \in \mathbb{R}^m$, $\gamma_t \in \mathbb{R}^m$ *is the parameter-vector of* $\widehat{f}$ *at*

the time t, and $D(\gamma_t, t)$ is a dynamic driver of $\widehat{f}$;
we call $\widehat{f}_{D(\gamma_t)} = \left\langle \widehat{f}, D(\gamma_t, t) \right\rangle$ a time-dependent problem *with respect to the time variable t. In this problem changes can be represented as changes in the parameter space and are controlled by the dynamic driver $D(\gamma_t, t)$.*

The inclusion of dynamic drivers and full-description form in the above definition distinguishes the definition from existing definitions of time-dependent problems. As discussed earlier, many existing definitions represent a time-dependent problem as a sequence of multiple static problems. These definitions might be ambiguous because there might be multiple ways to transfrom one static problem to another and hence it is not clear what type of dynamic the considered time-dependent problem has. The dynamic driver in Definition 4.7 represents the actual dynamic of the problem and hence it helps distinguish one time-dependent problem from another.

In some existing definitions (Rohlfshagen & Yao 2008, Bosman 2007), it has already been implied that changes in time-dependent problems can be represented as changes in the parameter space. In this chapter this concept will be formulated in a more detailed level and will be explicitly defined: *most common types of changes in time-dependent problems can be represented as changes in the parameter space if we can formulate the problem in a general enough full-description form.* This is true even in extreme cases where there is no correlation between the functions before and after a change. For example, a function-switching change from $f(x)$ at $t = 0$ to $g(x)$ at $t \geqslant 1$, $t \in \mathbb{N}^+$ can be expressed as $\widehat{f}(x) = a(t) f(x) + b(t) g(x)$ where $a(t)$ and $b(t)$ are two time-dependent parameters given by

$$\begin{cases} a(t) = 1 \text{ and } b(t) = 0 & \text{if } t = 0 \\ a(t) = 0 \text{ and } b(t) = 1 & \text{otherwise} \end{cases}$$

Dimensional changes, as found in some real-world systems, can also be represented as changes in the parameter given that the maximum number of variables is taken into account in the full-description form. For example, the function $\sum_{i=1}^{n} x_i^2$ with dimension n varies from 1 to 2 can be represented as the full-description form $\sum_{i=1}^{2} b_i(t) x_i^2$ with $b_i(t) \in \{0, 1\}$ depending on t.

Definition 4.8 (Time unit) *When a time-dependent problem is being solved, a* time unit*, or a unit for measuring time periods in the problem, represents the time durations needed to complete*

one function evaluation *of that problem.*[1] *The number of evaluations (or time units) that have been evaluated so far since we started solving the problem is measured by the variable* $\tau \in N^+$.

Definition 4.9 (Change step and frequency of change) *When a time-dependent problem is being solved, a* change step *represents the moment when the problem changes. The number of change steps that have occured so far in a time-dependent problem is measured by the variable* $t \in N^+$. *Obviously* t *is a time-dependent function of* τ- *the number of evaluations made so far since we started solving the problem;* $t(\tau) : \mathbb{N}^+ \longrightarrow \mathbb{N}^+$. *Its dynamic is controlled by a problem-specific* time-based dynamic driver*:*

$$t(\tau + 1) = D_T(t(\tau), \tau) \tag{4.10}$$

where $D_T(t(\tau), \tau)$ *is the problem-specific time-based dynamic driver. It decides the* frequency of change *of the problem and can be described as follows:*

$$\begin{cases} D_T(t(\tau), \tau) = t(\tau) + 1 & \text{when a change occurs} \\ D_T(t(\tau), \tau) = t(\tau) & \text{otherwise} \end{cases} \tag{4.11}$$

Definition 4.12 (Optimisation algorithms and dynamic solutions) *Given a time-dependent problem* $\widehat{f}_{D(\gamma_t)} = \left\langle \widehat{f}, D(\gamma_t, t) \right\rangle$ *at the change step* t *(see Definition 4.7) and a set* P_t *of* k_t *solutions* $\mathbf{x}_1, ..., \mathbf{x}_{k_t} \in S_t$ *where* $S_t \subseteq \mathbb{R}^d$ *is the search space*[2]*,*
an optimisation algorithm G *to solve* $\widehat{f}_{D(\gamma_t)}$ *can be seen as a mapping*

$$G_t : \mathbb{R}^{d \times k_t} \rightarrow \mathbb{R}^{d \times k_{t+1}} \tag{4.13}$$

capable of producing a solution set P_{t+1} *of* k_{t+1} *optimised solutions* $\mathbf{x}_1^G, ..., \mathbf{x}_{k_{t+1}}^G$ *at the next change step* $t + 1$*:*

$$P_{t+1} = G_t(P_t) \ . \tag{4.14}$$

Generally, at a change step $t^e \in \mathbb{N}^+$ *the set of* dynamic solutions $X_{f_t}^{G_{[t^b, t^e]}}$ *that we get by applying an algorithm* G *to solve* $\widehat{f}_{D(\gamma_t)}$ *with a given initial population* $P_{t^b - 1}$ *during the period* $\left[t^b, t^e\right]$,

[1] As mentioned in (Bäck 1998) and (Rohlfshagen & Yao 2008), from the perspective of optimisation algorithms time is descrete and the smallest time unit is one function evaluation.

[2] Here we are considering search spaces $\subseteq \mathbb{R}^d$. However the definition can be generalized for other non-numerical encoding algorithms by replacing $\mathbb{R}^d$ with the appropriate encoding space.

$t^b \geqslant 1$ *is given by:*

$$X_{f_t}^{G_{[t^b,t^e]}} = \bigcup_{t=t^b}^{t^e} P_t = \bigcup_{t=t^b}^{t^e} G_t\left(P_{t-1}\right) \ . \tag{4.15}$$

In real-world time-dependent problems, some time-dependent rules that change the problems' parameters may have the *time-linkage* feature, i.e. they take solutions found by the algorithm up to the current time step as their parameters. In such cases, the time-linkage dynamic rules can be defined mathematically as follows.

Definition 4.16 (Time-linkage dynamic driver) *Given a tuple* $\left\langle \widehat{f}, \gamma_t, t, X_{\widehat{f}}^{G_{[1,t]}} \right\rangle$ *where* t *is a time variable,* $\widehat{f}$ *is a full-description form of the set of functions* $F = \{f_1(x), ..., f_n(x)\}$ *with respect to the set of* m*-element vectors* $\{\mathbf{c}_1, ..., \mathbf{c}_n\}, \mathbf{c}_i \in \mathbb{R}^m$*,* $\gamma_t \in \mathbb{R}^m$ *is an* m*-element vector containing all* m *parameters of* $\widehat{f}$ *at the time* t*, ; and* $X_{\widehat{f}}^{G_{[1,t]}}$ *is a set of* k d*-dimensional solutions achieved by applying an algorithm* G *to solve* $\widehat{f}$ *during the period* $[1, t]$*;*
we call a mapping $D\left(\gamma_t, X_{\widehat{f}}^{G_{[1,t]}}, t\right) : \mathbb{R}^m \times \mathbb{R}^{d\times k} \times \mathbb{N}^+ \longrightarrow \mathbb{R}^m$ *a* time-linkage dynamic driver *of* $\widehat{f}$ *if*

$$\gamma_{t+1} = D\left(\gamma_t, X_{\widehat{f}}^{G_{[1,t]}}, t\right) \ \in \{\mathbf{c}_1, ..., \mathbf{c}_n\} \, \forall t \in \mathbb{N}^+ \tag{4.17}$$

and

$$\gamma_{t+1} \text{ is used as the set of parameters of } \widehat{f} \text{ at the time } t+1$$

There are cases where $X_{\widehat{f}}^{G_{[1,t]}}$ does not have any influence on the future of $\widehat{f}$. In these cases $D\left(\gamma_t, X_{\widehat{f}}^{G_{[1,t]}}, t\right)$ becomes a regular dynamic driver with no time-linkage feature.

Definition 4.18 (Dynamic optimisation problem) *Given a tuple* $\left\langle \widehat{f}, \widehat{C}, D_P, D_D, D_T, G \right\rangle$*, a* dynamic optimisation problem *in the period* $\left[1, \tau^{end}\right]$ *function evaluations,* $\tau^{end} \in \mathbb{N}^+$ *can be defined as*

$$\text{optimise} \left\{ \sum_{\tau=1}^{\tau^{end}} \widehat{f}_{\gamma\left(t_\tau, X_{\widehat{f}}^{G_{[1,t]}}\right)}\left(\mathbf{x}_t\right) \right\} \tag{4.19}$$

subject to $\widehat{C}^{i=1:k\in N^+}_{\gamma\left(t_\tau, X^{G_{[1,t]}}_{\widehat{f}}\right)}(\mathbf{x}_t, t_\tau) \leqslant 0$; *and* $\mathbf{l}\left(t_\tau, X^{G_{[1,t]}}_{\widehat{f}}\right) \leqslant \mathbf{x} \leqslant \mathbf{u}\left(t_\tau, X^{G_{[1,t]}}_{\widehat{f}}\right)$ *where*

$\widehat{f}$ *is the full-description form of the objective function*

$\widehat{C}^1 ... \widehat{C}^k$ *are the full-description forms of* k *dynamic constraints*[3]

D_P *is the dynamic driver for parameters in objective and constraint (see below)*

D_D *is the dynamic driver for domain constraints (see below)*

D_T *is the dynamic driver for times and frequency of changes(Equation 4.11)*

G *is the algorithm used to solve the problem*

$\tau \in \left[1, \tau^{end}\right] \cap \mathbb{N}$ *is the number of function evaluations done so far*

t_τ, *or* $t(\tau) \in \mathbb{N}^+$ *is the current change step;* $t(\tau)$ *is controlled by* D_T *(Equation 4.11)*

$X^{G_{[1,t]}}_{\widehat{f}}$ *is the set of solutions achieved by applying the algorithm* G *to solve* $\widehat{f}$ *during* $[1,t]$

$\gamma_{t_\tau} \in \mathbb{R}^p$ *is the time-dependant parameters of* $\widehat{f}$ *and* $\widehat{C}^i$; $\gamma_{t_\tau+1} = D_P\left(\gamma_{t_\tau}, X^{G_{[1,t]}}_{\widehat{f}}, t\right)$

$\mathbf{l}(t_\tau), \mathbf{u}(t_\tau) \in \mathbb{R}^n$ *are domain constraints;* $\left\{\begin{matrix}\mathbf{l}(t_\tau+1)=D_D\left(\mathbf{l}(t_\tau), X^{G_{[1,t]}}_{\widehat{f}}, t_\tau\right)\\ \mathbf{u}(t_\tau+1)=D_D\left(\mathbf{u}(t_\tau), X^{G_{[1,t]}}_{\widehat{f}}, t_\tau\right)\end{matrix}\right.$

□

The new definition brings us some advantages. First, with the introduction of the *change step*, the *optimisation algorithm* and the *dynamic solutions* produced by the algorithm at each change step, the definition clearly defines DOPs as *time-dependent problems that are solved online in a dynamic way*, and hence distinguishes DOPs from other time-dependent problems. Second, we can now classify DOPs based on three distinguished components: the *full-description forms*, the *dynamic drivers*, and the *algorithm*. This separation facilitates us in characterising DOPs and evaluating the impact of each components on the difficulty of the problems. Third, the definition supports an important feature of dynamic time-linkage problems that has not been fully considered before: *algorithm-dependent*. Fourth, the definition encapsulates different aspects of DOPs such as dynamic rules, change frequencies, changes in constraints, changes in domain range, changes in objective functions in details.

[3] These also include equality constraints because any equality constraint $c(x) = 0$ can be transformed into an inequality $|c(x)| - \varepsilon \leqslant 0$ with a small value ε.

4.3 Summary of contributions

The contribution of this chapter is to provide a new definition framework, which is expected to help to close some gaps that existing formal definitions of DOPs have not addressed. The contributions can be categories as follows:

1. *Distinguishing DOPs* from other time-dependent problems.

2. *Taking into account the algorithm-dependence property.*

3. *Covering many aspects of a DOP that has not been considered in details in previous DOP definitions*: time unit, change step, frequency of changes, changes in constraints and changes in domain range.

4. *Representing DOPs based on three distinguished components: the full-description forms, the dynamic drivers, and the algorithm* to make it easier to study the behaviour of DOPs. The idea of separating the dynamic drivers from the static description forms will also help in generating DOP benchmarks from existing well-studied static benchmarks. In the next chapter, we will use this principle to design a set of benchmark problems for dynamic constrained optimisation. This principle was also used to generate the dynamics for the dynamic benchmark problems in the CEC'2009 Competition on Dynamic Optimization (Li *et al.* 2008).

Chapter 5

Analysing the difficulties of existing dynamic optimisation and constraint handling algorithms in solving DCOPs

The research in this chapter aims to answer some open questions about the characteristics, difficulty and solution methods of a very common class of problems - dynamic constrained optimisation problems (DCOPs). DCOPs are constrained optimisation problems that have two properties: First, the objective functions, the constraints, or both, may change over time, and second, the changes are taken into account in the optimisation process[1]. The review in Chapter 3 has shown that a majority of the surveyed real-world dynamic problems are DCOPs. However, there are few studies on continuous dynamic constrained optimisation (DCO). Specifically, there is little research on whether current numerical dynamic optimisation (DO) algorithms and numerical constraint handling (CH) algorithms would work well in DCOPs. There is also no numerical dynamic constrained benchmark problem that reflects the common characteristics of DCOPs. Existing studies in continuous DO only focus on the unconstrained or domain constraint dynamic cases (which in this thesis I regard both as "unconstrained" problems). Likewise, existing research in CH only focuses on the stationary constrained problems.

[1] This definition is derived from the (more general) definition of dynamic optimisation problems in (Jin & Branke 2005, section V).

This lack of attention on DCOPs in the continuous domain raises some important research questions. First, what are the essential characteristics of DCOPs? Second, how well would existing DO and CH strategies perform in DCOPs if most of them are designed for and tested in either unconstrained dynamic problems or stationary constrained problems only? And why do they work well or not well? Third, how can we evaluate if an algorithm works well or not in DCOPs? And fourth, what are the requirements for a "good" algorithm to effectively solve DCOPs?

As a large number of real-world applications are DCOPs, I believe that finding the answers to the questions above is essential. This is because such answers would help us to have more understanding about the practical issues of the problems and to solve this class of problems more effectively.

This chapter contributes to the task of finding such answers. First, in section 5.1, I will identify the special characteristics of DCOPs from real-world references. I will also discuss how these characteristics make DCOPs different from unconstrained dynamic optimisation problems (DOPs). Then in Section 5.2, I will firstly review related literature about continuous DCO benchmark problems, and identify the gaps between them and common DCOPs. Then I will propose a new method to generate general dynamic benchmark problems. That method will then be used to introduce a new set of DCO benchmark problems, which are able to represent the characteristics identified in section 5.1. In the next section (Section 5.3), I will investigate the possibility of solving DCOPs using some representative DO strategies. Experimental analyses about the strengths and weaknesses of existing DO strategies, and the effect of the mentioned characteristics on each strategy will also be undertaken. Based on the experimental results, I will then suggest a list of requirements that a DO algorithm should meet to solve DCOPs effectively. In section 5.4, similar literature reviews and experimental analyses about the possibility of solving DCOPs will again be carried out, but now under the perspective of existing CH strategies. Similar to the previous section, in this section I will also suggest a list of possible requirements that a CH algorithm should meet to solve DCOPs effectively. The chapter finishes with Section 5.5, where some conclusions and future directions will be discussed.

5.1 The common characteristics of dynamic constrained problems

The presence of constraints in DCOPs make them very different from the unconstrained or domain constraint problems (problems where the only constraints are the bounds of decision variables) considered in academic research (in this chapter we conventionally name both unconstrained problems and domain constraint problems as "unconstrained problems"). Different from the unconstrained problems where there is only one objective function with no constraint or with domain constraints only, a DCOP is a combination of the objective function and one or many constraint functions, in which at least one of these objective/constraint functions is dynamic. In real-world DCOPs the objective function and constraint functions can be combined in three different types. The first type of combination is the case where both the objective function and the constraints are dynamic, as in scheduling/resource allocation problems (Andrews & Tuson 2005), aerodynamic/structural design problems (Padula *et al.* 2006), or in many optimal control problems (Schlegel & Marquardt 2006, Wang & Wineberg 2006, Prata *et al.* 2006). The second type of combination is the case where only the objective function is dynamic while the constraints are static, for example the document stream modelling problem (Araujo & Merelo 2007), the evolvable hardware designing problem (Tawdross *et al.* 2006) or the optimal control problem of fermentation processes (Rocha *et al.* 2005). In the third type of combination, the objective function is static and the constraints are dynamic, as can be seen in the hydrothermal scheduling problem (Deb *et al.* 2007), the cargo movement problem (Ioannou *et al.* 2002) and the ship scheduling problem (Mertens *et al.* 2006). In all three types of combination, the presence of infeasible areas can affect the way the global optimum moves, or appears after each change. This leads to some special characteristics which cannot be found in the unconstrained cases and fixed constrained cases.

The first special characteristic is the fact that the dynamic of constraints can lead to changes in the shape/size/structure of the feasible/infeasible areas. Examples of this behaviour can be found in the problems with dynamic constraints from the real-world applications mentioned above (Andrews & Tuson 2005, Schlegel & Marquardt 2006, Wang & Wineberg 2006, Prata *et al.* 2006, Deb *et al.* 2007, Ioannou *et al.* 2002, Padula *et al.* 2006, Mertens *et al.* 2006).

The second special characteristic is the fact that a dynamic objective function might cause

the global optima to switch from one disconnected feasible region to another in problems with disconnected feasible regions. Disconnected feasible regions are very common in real-world constrained problems, especially the scheduling problems. Some examples are the examination timetabling problems (Thompson & Dowsland 1998, Thompson & Dowsland 1996), the nurse rostering problems (Dowsland 1998, Aickelin & Dowsland 2000), the enterprise-driven multilevel product design problems (Kim 2006), and the video-based motion capture problems (Gleicher & Ferrier 2002). Bartusch et al. (Bartusch *et al.* 1988) have mathematically shown that the feasible regions of project scheduling problems with general temporal and resource constraints are generally disconnected. In such problems, the global optima might switch from one disconnected feasible region to another if the objective function is dynamic and the constraints are fixed. In this case, because the number, locations and sizes of disconnected feasible regions are still the same, after a change the new global optimum can only either (1) stay in the previous disconnected region or (2) move to another disconnected region.

The third special characteristic of DCOPs is that in problems with fixed objective functions and dynamic constraints, the changing infeasible areas might expose new, better global optima without changing the existing optima. One example can be found in the Dynamic 0-1 Knapsack Problem: significantly decreasing the weight of a high-value object that is not included in the current global optimal solution might create a new global optimal solution without changing the value of the existing one. Similarly, significantly increasing the capacity of the knapsack might also create a new global optimum without changing the value of the existing optimum. As shown later, such types of changes might make the problem difficult to some of the existing dynamic optimisation strategies because these strategies only focus on tracking the existing global optimum.

In addition to the three special characteristics above, DCOPs might also have the common characteristics of constrained problems as global optima in the boundaries of feasible regions, global optima in search boundary, and multiple disconnected feasible regions. Similar to the three special characteristics of DCOPs above, these characteristics also are widely regarded as being common in real-world applications. Another common characteristic of DCOPs from real-world applications, as I found in my review in Chapter 3, is that the dynamic of the objective functions or constraints usually follow some time-dependant functions or rules rather than just behaving randomly.

Although the characteristics mentioned above are common in real-world applications in both combinatorial and continuous domains, they have only been considered mostly in the combinatorial domain. Particularly, in the continuous domain these characteristics have never been captured in existing standard DO benchmark problems, as shown by the list of available benchmark generators in Section 2.3. In the next section I will describe a set of DCOP benchmark problems to overcome this issue.

5.2 A set of real-valued benchmark problems to simulate DCOPs characteristics

5.2.1 Related literature

In the continuous domain, to the best of my knowledge, besides this research there is no existing continuous benchmark that fully reflects the characteristics of DCOPs listed in Section 5.1. Among the existing continuous benchmarks, there are only two recent studies that closely relate to dynamic constraint problems (many existing continuous dynamic benchmark do have domain constraints, but as mentioned earlier in this chapter we consider domain constraint problems "unconstrained problems"). The first study is the recent paper reported by Liu (2008*a*) in which two test problems (DCT2 and DCT3) are proposed. These problems are two simple unimodal constrained problems which take the time variable t as their only time-dependant parameter. Because of that, the dynamic is created by the increase over time of t.

Although these two problems are indeed dynamic constrained problems, they have some important disadvantages which prevent us from using them to capture/simulate the properties of DCOPs mentioned in Section 5.1. First, it is impossible to realistically simulate the dynamic rules/functions from DO applications using these problems because they only capture a simple linear change, while many real-world applications may have different types of non-linear changes.. Second, there is no evidence that the two problems can convey any common properties of DCOPs such as optima in the boundary; disconnected feasible regions; and moving constraints exposing optima. Third, the two problems do not reflect common situations like dynamic objective + fixed constraints or fixed objective + dynamic constraints. Finally, the small number of test functions in (Liu 2008*a*) (two functions) might not be enough to evaluate algorithms under different situations. In order to evaluate the performance of algorithms in DCOPs, a large variety of test

problems should be used to study the strengths and weakness of the tested algorithms.

The second study, which was just published at the time this thesis was being prepared for submission, is the research by Richter (2010). In this work, a dynamic constrained benchmark problem was proposed by combining an existing "field of cones on a zero plane" dynamic fitness function with four dynamic norm-based constraints with square/diamond/sphere-like shapes (see Figure 2 in (Richter 2010)). The framework used to generate this benchmark problem is highly configurable because it allows designers to control the geometrical shapes, positions and sizes of constrained areas as well as the position and shapes of the fitness landscape. We believe that with further extensions and careful designs that framework can be used to generate different benchmark problems that are able to capture some of the common characteristics of DCOPs mentioned in Section 5.1. The current single benchmark problem generated by the framework in (Richter 2010), however, was designed for a different purpose and hence does not serve the purpose of simulating the properties mentioned in Section 5.1 yet. For example, the problem might not be able to simulate such properties of common DCOPs such as optima in boundary; disconnected feasible regions; and moving constraints exposing optima in a controllable way. In addition, there is only one single benchmark problem and hence it might be difficult to use the problem to evaluate the performance of algorithms under different situations.

Another research which somehow involves changing constraints is the work of Jin *et al.* (2010). However, the study in (Jin *et al.* 2010) is very different from the work in this chapter because in (Jin *et al.* 2010) the considered constrained problems are stationary and the authors only change the original constraint functions purposely during the optimisation process to make it easier for the algorithm to find its way to the global optimum. In other words, in (Jin *et al.* 2010) the actual constrained functions are not time-dependent and hence the problems are not DOPs.

The lack of a set of benchmark problems to capture the common characteristics of DCOPs (mentioned in Section 5.1) makes it difficult to evaluate how well existing DO algorithms would work in DCOPs, as they have been designed and tested in unconstrained/domain constrained problems only. This lack of an appropriate benchmark would also pose some difficulties in designing/developing new algorithms specialising for DCOPs because algorithm designers would not be able to know if their algorithms work well in DCOPs. Given the fact that a majority of recent real-world dynamic optimisation problems are constrained problems as shown in Subsection 3.3.1, the lack of an appropriate set of benchmark problems can be considered an important

gap in current dynamic optimisation research.

This gap motivates me to develop a full set of benchmark problems to capture the special characteristics of DCOPs. Some initial results involving five benchmark problems, which were able to capture some characteristics of DCOPs, have been reported in an earlier study (Nguyen & Yao 2009*a*). In this chapter I will extend this framework to develop full sets of benchmark problems, which are able to capture all characteristics mentioned in the previous section. The problems can also be incorporated with different types of dynamic rules to better simulate different dynamic constrained applications. Two sets of benchmark problems, one with multimodal, scalable objective functions and one with unimodal objective functions, have been developed for this research. In this chapter I will discuss in detail the benchmark set with unimodal objective function (in spite of the fact that the objective function is unimodal, many problems in the set still have multiple optima due to the constraints). I choose the unimodal benchmark for the analyses in this chapter because they are less complicated and hence can facilitate us better in analysing the behaviours of algorithms. Details of the multimodal, scalable set can be found in our technical report (Nguyen 2008*a*).

5.2.2 Generating dynamic constrained benchmark problems

In chapter 4 we have discussed that most dynamic problems can be represented as a combination of a static full-description function form and a dynamic driver, which represent changes in the parameter space. Here we can use this procedure to create new dynamic benchmark problems by combining (a) an existing static benchmark problem (which represent the full-description function forms) $f_P(x)$ where $P = \{p_1, ...p_k\}$ is the set of static parameters with (b) some time-dependent parameters $p_i(t)$ (which represents the dynamic drivers). The resulting dynamic benchmark problem $f_{P_t}(x, t)$ is generated by replacing each static parameter $p_i \in P$ with a corresponding time-dependent expression $p_i(t)$. The dynamic of the dynamic problem then depends on how $p_i(t)$ varies over time. We can use any type of dynamic rule from practical problems to represent $p_i(t)$ and hence we can create any type of dynamic problem we want.

Detail of applying the idea above to generating a comprehensive set of DCOP benchmark will be described in the next subsection.

5.2.3 A dynamic constrained benchmark set

Using the new procedure described in the previous subsection, in this chapter I introduce a set of benchmark problems named G24. The set contains 18 problems, each with a unimodal objective function (despite that the objective function is unimodal, many problems in the set still have multiple feasible optima due to the presence of multiple disconnected feasible regions). Most problems in the set are modified from one of the static functions proposed in (Floudas *et al.* 1999). Some others, however, have entirely new constraint functions (G24_6a, G24_6b, G24_6c and G24_6d) or new objective functions (G24_8a and G24_8b) specifically designed in our research to simulate some special characteristics of DCOPs. The objective functions and constraint functions of all problems are then combined with our newly proposed dynamic rules, which are specifically designed to reflect the different properties of DCOPs as mentioned in Section 5.1.

The general form for each problem in the G24 set is as follows:

$$\begin{aligned} \text{minimise} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) \leqslant 0,\ g_i(\mathbf{x}) \in G,\ i = 1, .., n \end{aligned}$$

where the objective function $f(\mathbf{x})$ can be one of the full-description function forms set out in equation (5.1), each constraint $g_i(\mathbf{x})$ can be one of the full-description function forms given in equation (5.2), and G is the set of n constraint functions for that particular benchmark problem in the G24 set. The detailed description of $f(\mathbf{x})$ and $g_i(\mathbf{x})$ for each problem in the G24 benchmark set are described in Table 5.1 (page 103) and Table 5.2 (page 103).

Equation (5.1) below describes the general function forms from which I will develop the objective functions for each benchmark problem in G24 set. Of these function forms, $f^{(2)}$is used to design the objective function for G24_8a and G24_8b, and $f^{(1)}$ is used to design the objective functions for all other problems. $f^{(1)}$ is modified from a static function proposed in (Floudas *et al.* 1999) and $f^{(2)}$is a newly designed function. It should be noted that in the expression of $f^{(2)}$, the (-3) factor was used to scale the function values to the same range as used in $f^{(1)}$ and the square roots in $f^{(2)}$ were used to make the basin of attraction become narrower and hence

Table 5.1: The objective function form of each benchmark problem in the G24 benchmark set

Benchmark problem	objective function
G24_8a & G24_8b	$f(x) = f^{(2)}$
All other problems	$f(x) = f^{(1)}$

Table 5.2: The set of constraint function forms for each benchmark problem in the G24 benchmark set

Benchmark problem	Set G of constraints
G24_u; G24_uf; G24_2u; G24_8a	$G = \{\emptyset\}$
G24_6a	$G = \left\{g^{(3)}, g^{(6)}\right\}$
G24_6b	$G = \left\{g^{(3)}\right\}$
G24_6c	$G = \left\{g^{(3)}, g^{(4)}\right\}$
G24_6d	$G = \left\{g^{(5)}, g^{(6)}\right\}$
All other problems	$G = \left\{g^{(1)}, g^{(2)}\right\}$

more difficult to find the global optimum (the more square roots the narrower the basin).

$$\begin{aligned} f^{(1)} = & \quad -\left(X_1\left(x_1,t\right) + X_2\left(x_2,t\right)\right) \\ f^{(2)} = & \quad -3\exp\left(-\sqrt{\sqrt{\left(X_1\left(x_1,t\right)\right)^2 + \left(X_2\left(x_2,t\right)\right)^2}}\right) \end{aligned} \tag{5.1}$$

where

$$X_i\left(x,t\right) = p_i\left(t\right)\left(x + q_i\left(t\right)\right); 0 \leqslant x_1 \leqslant 3; 0 \leqslant x_2 \leqslant 4$$

with $p_i\left(t\right)$ and $q_i\left(t\right)$ $(i = 1, 2)$ as the dynamic parameters, which determine how the dynamic objective function of each benchmark problem changes over time. Each benchmark problem may have a different mathematical expression for $p_i\left(t\right)$ and $q_i\left(t\right)$. It should be noted that although many benchmark problems share the same general full-description function form in equation (5.1), their individual expressions for $p_i\left(t\right)$ and $q_i\left(t\right)$ make their actual dynamic objective functions very different. The individual expressions of $p_i\left(t\right)$ and $q_i\left(t\right)$ for each benchmark function are described in Table 5.3 (page 105).

Equation (5.2) below describes the general function forms from which I will develop the constraint functions for each benchmark problem in the G24 set. Of these function forms, $g^{(1)}$and $g^{(2)}$are modified from two static functions proposed in (Floudas *et al.* 1999) and $g^{(3)}, g^{(4)}$and $g^{(5)}$ are newly designed functions. Each benchmark problem may use only a subset of constraint functions from equation (5.2). Details of which constraint function is used in which benchmark

problem are given in Table 5.2 (page 103).

$$
\begin{aligned}
g^{(1)} &= -2Y_1(x_1,t)^4 + 8Y_1(x_1,t)^3 - 8Y_1(x_1,t)^2 + Y_2(x_2,t) - 2 \qquad (5.2)\\
g^{(2)} &= -4Y_1(x_1,t)^4 + 32Y_1(x_1,t)^3 - 88Y_1(x_1,t)^2 + 96Y_1(x_1,t) + Y_2(x_2,t) - 36\\
g^{(3)} &= 2Y_1(x_1,t) + 3Y_2(x_2,t) - 9\\
g^{(4)} &= \begin{cases} -1 & \text{if } (0 \leqslant Y_1(x_1,t) \leqslant 1) \text{ or } (2 \leqslant Y_1(x_1,t) \leqslant 3)\\ 1 & \text{otherwise} \end{cases}\\
g^{(5)} &= \begin{cases} -1 & \text{if } (0 \leqslant Y_1(x_1,t) \leqslant 0.5) \text{ or } (2 \leqslant Y_1(x_1,t) \leqslant 2.5)\\ 1 & \text{otherwise} \end{cases}\\
g^{(6)} &= \begin{cases} -1 & \text{if } [(0 \leqslant Y_1(x_1,t) \leqslant 1) \text{ and } (2 \leqslant Y_2(x_2,t) \leqslant 3)] \text{ or } (2 \leqslant Y_1(x_1,t) \leqslant 3)\\ 1 & \text{otherwise} \end{cases}
\end{aligned}
$$

where

$$Y_i(x,t) = r_i(t)(x + s_i(t)); 0 \leqslant x_1 \leqslant 3; 0 \leqslant x_2 \leqslant 4$$

with $r_i(t)$ and $s_i(t)$ $(i = 1, 2)$ as the dynamic parameters, which determine how the constraint functions of each benchmark problem change over time. Each benchmark problem may have a different mathematical expression for $r_i(t)$ and $s_i(t)$. It should be noted that although the constraint functions of many benchmark problems might share the same general full-description function form in equation (5.2), their individual expressions for $r_i(t)$ and $s_i(t)$ make their actual dynamic constraint functions very different. The individual expression of $r_i(t)$ and $s_i(t)$ for each benchmark function are described in Table 5.3 (page 105).

To design the test problems, I follow two design guidelines. First, although we can create any arbitrary number of test problems based on the basic function forms given in Table 5.1 (page 103) and Table 5.2 (page 103), we are only interested in creating problems that can simulate the properties of common DCOPs as mentioned in Section 5.1 because they have not been captured in existing continuous dynamic benchmark problems. These problems will be used as the benchmark to answer the question of whether existing dynamic optimisation strategies and constraint handling strategies would work well in DCOPs.

Second, to make it easy to analyse the effect of each characteristic of DCOPs on the perfor-

Table 5.3: Dynamic parameters for all test problems in the benchmark set G24. Each dynamic parameter is a time-dependant rule/function which governs the way the problems change

Prob	Parameter settings
G24_u	$p_1(t) = \sin\left(k\pi t + \frac{\pi}{2}\right); p_2(t) = 1; q_i(t) = 0$
G24_1	$p_2(t) = r_i(t) = 1;\ q_i(t) = s_i(t) = 0$ $p_1(t) = \sin\left(k\pi t + \frac{\pi}{2}\right)$
G24_f	$p_i(t) = r_i(t) = 1;\ q_i(t) = s_i(t) = 0$
G24_uf	$p_i(t) = 1;\ q_i(t) = 1$
G24_2	if $(t \bmod 2 = 0)$ $\begin{cases} p_1(t)=\sin\left(\frac{k\pi t}{2}+\frac{\pi}{2}\right) \\ p_2(t)=\begin{cases} p_2(t-1) \text{ if } t>0 \\ p_2(0)=0 \text{ if } t=0 \end{cases} \end{cases}$ if $(t \bmod 2 \neq 0)$ $\begin{cases} p_1(t)=\sin\left(\frac{k\pi t}{2}+\frac{\pi}{2}\right) \\ p_2(t)=\sin\left(\frac{k\pi(t-1)}{2}+\frac{\pi}{2}\right) \end{cases}$ $q_i(t) = s_i(t) = 0;\ r_i(t) = 1$
G24_2u	if $(t \bmod 2 = 0)$ $\begin{cases} p_1(t)=\sin\left(\frac{k\pi t}{2}+\frac{\pi}{2}\right) \\ p_2(t)=\begin{cases} p_2(t-1) \text{ if } t>0 \\ p_2(0)=0 \text{ if } t=0 \end{cases} \end{cases}$ if $(t \bmod 2 \neq 0)$ $\begin{cases} p_1(t)=\sin\left(\frac{k\pi t}{2}+\frac{\pi}{2}\right) \\ p_2(t)=\sin\left(\frac{k\pi(t-1)}{2}+\frac{\pi}{2}\right) \end{cases}$ $q_i(t) = 0$
G24_3	$p_i(t) = r_i(t) = 1; q_i(t) = s_1(t) = 0$ $s_2(t) = 2 + t.\frac{x_2\max - x_2\min}{S}$
G24_3b	$p_1(t) = \sin\left(k\pi t + \frac{\pi}{2}\right);\quad p_2(t) = 1$ $q_i(t) = s_1(t) = 0;\ r_i(t) = 1;$
	$s_2(t) = 2 + t.\frac{x_2\max - x_2\min}{S}$
G24_3f	$p_i(t) = r_i(t) = 1; q_i(t) = s_1(t) = 0; s_2(t) = 2$
G24_4	$p_2(t) = r_i(t) = 1;\ q_i(t) = s_1(t) = 0$ $p_1(t) = \sin\left(k\pi t + \frac{\pi}{2}\right); s_2(t) = t.\frac{x_2\max - x_2\min}{S}$
G24_5	if $(t \bmod 2 = 0)$ $\begin{cases} p_1(t)=\sin\left(\frac{k\pi t}{2}+\frac{\pi}{2}\right) \\ p_2(t)=\begin{cases} p_2(t-1) \text{ if } t>0 \\ p_2(0) \text{ if } t=0 \end{cases} \end{cases}$ if $(t \bmod 2 \neq 0)$ $\begin{cases} p_1(t)=\sin\left(\frac{k\pi t}{2}+\frac{\pi}{2}\right) \\ p_2(t)=\sin\left(\frac{k\pi(t-1)}{2}+\frac{\pi}{2}\right) \end{cases}$ $q_i(t) = s_1(t) = 0;\ r_i(t) = 1;$
	$s_2(t) = t.\frac{x_2\max - x_2\min}{S}$
G24_6a/b/c/d	$p_1(t) = \sin\left(\pi t + \frac{\pi}{2}\right); p_2(t) = 1;$
	$q_i(t) = s_i(t) = 0; r_i(t) = 1$
G24_7	$p_i(t) = r_i(t) = 1;\ q_i(t) = s_1(t) = 0;$
	$s_2(t) = t.\frac{x_2\max - x_2\min}{S}$
G24_8a	$p_i(t) = -1; q_1(t) = -\left(c_1 + r_a.\cos(k\pi t)\right)$ $q_2(t) = -\left(c_2 + r_a.\sin(k\pi t)\right);$
G24_8b	$p_i(t) = -1; q_1(t) = -\left(c_1 + r_a.\cos(k\pi t)\right)$ $q_2(t) = -\left(c_2 + r_a.\sin(k\pi t)\right); r_i(t) = 1;\ s_i(t) = 0$
k	k determines the severity of function changes. $k = 1 \sim$large; $k = 0.5 \sim$ medium; $k = 0.25 \sim$ small
S	S determines the severity of constraint changes $S = 10 \sim$large; $S = 20 \sim$ medium; $S = 50 \sim$ small
c_1, c_2, r_a (G24_8a/b only)	$c_1 = 1.4706; c_2 = 3.442; r_a = 0.859$
i	i is the variable index, $i = 1, 2$

mance of the tested algorithms, there should always be a pair of problems for each characteristic. The two problems in this pair should be almost identical except for that one has a particular characteristic (e.g. fixed constraints) and the other does not. By comparing the performance of an algorithm on one problem with its performance on the other problem in the pair, we will be able to analyse whether the considered characteristic has any effect on the tested algorithm and to what extent is the effect significant.

Based on the two guidelines above, I have designed 18 different sets of dynamic parameters to create 18 different test problems for the dynamic constrained benchmark set G24 (Table 5.3, page 105). Each test problem is able to capture one or several characteristics of DCOPs, as shown in table 5.4 (page 107). In addition, the problems and their relationships are carefully designed so that they can be arranged in 21 pairs, of which each pair is a different test case to test a single characteristic of DCOPs (Table 5.5, page 108). An example showing the landscape in different change steps of one problem in the set (G24_4) can be seen in Figure 5.1 (page 106).

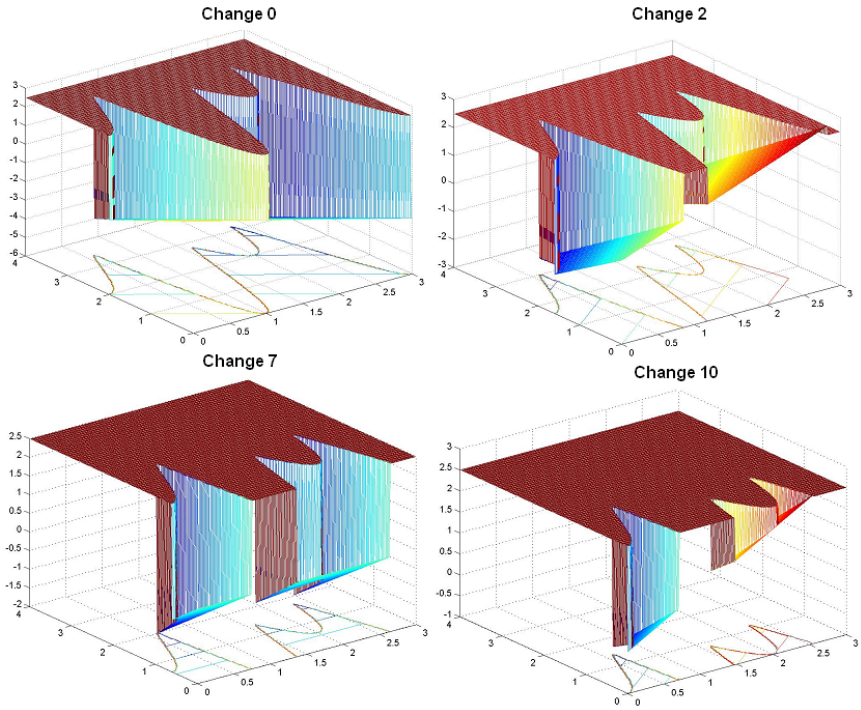

Figure 5.1: This figure illustrates the feasible search landscapes of one problem of G24 - the G24_4, at four different change periods: before the first change and at the second/seventh/tenth changes, respectively. The z axis represents objective values. The shaded areas in the figures are the projections of infeasible regions to the plane $z = 2.5$ (for illustration purpose). We can see that both the dynamic objective function and the constraints change over time. The size and shape of the feasible areas, and the number of disconnected regions also change accordingly.

Table 5.4: Properties of each test problem in the G24 benchmark set

Problem	ObjFunc	Constr	DFR	SwO	bNAO	OICB	OISB	Path
G24_u	Dynamic	NoC	1	No	No	No	Yes	N/A
G24_1	Dynamic	Fixed	2	Yes	No	Yes	No	N/A
G24_f	Fixed	Fixed	2	No	No	Yes	No	N/A
G24_uf	Fixed	NoC	1	No	No	No	Yes	N/A
G24_2*	Dynamic	Fixed	2	Yes	No	Yes&No	Yes&No	N/A
G24_2u	Dynamic	NoC	1	No	No	No	Yes	N/A
G24_3	Fixed	Dynamic	2-3	No	Yes	Yes	No	N/A
G24_3b	Dynamic	Dynamic	2-3	Yes	No	Yes	No	N/A
G24_3f	Fixed	Fixed	1	No	No	Yes	No	N/A
G24_4	Dynamic	Dynamic	2-3	Yes	No	Yes	No	N/A
G24_5*	Dynamic	Dynamic	2-3	Yes	No	Yes&No	Yes&No	N/A
G24_6a	Dynamic	Fixed	2	Yes	No	No	Yes	Hard
G24_6b	Dynamic	NoC	1	No	No	No	Yes	N/A
G24_6c	Dynamic	Fixed	2	Yes	No	No	Yes	Easy
G24_6d	Dynamic	Fixed	2	Yes	No	No	Yes	Hard
G24_7	Fixed	Dynamic	2	No	No	Yes	No	N/A
G24_8a	Dynamic	NoC	1	No	No	No	No	N/A
G24_8b	Dynamic	Fixed	2	Yes	No	Yes	No	N/A

DFR	number of Disconnected Feasible Regions
SwO	Switched global Optimum between disconnected regions
bNAO	better Newly Appear Optimum without changing existing ones
OICB	global Optimum is In the Constraint Boundary
OISB	global Optimum is In the Search Boundary
Path	Indicate if it is easy or difficult to use mutation to travel between feasible regions
Dynamic	The function is dynamic
Fixed	There is no change
NoC	There is no constraint
*	In some change periods, the landscape either is a plateau or contains infinite number of optima and all optima (including the existing optimum) lie in a line parallel to one of the axes

5.3 Difficulties of applying current dynamic optimisation strategies directly to solving DCOPs - an analysis

Table 5.5: This table shows the 21 test cases (pairs) to be used in this chapter. It should be noted that there is one situation where two test cases (10 and 14) use the same pair of problems. However, there is no redundancy because the two test cases are used to analyse different characteristics.

Static problems: Unconstrained vs Fixed constraints			
1	G24_uf (fF, noC)	vs	G24_f (fF, fC)
Fixed objectives vs Dynamic objectives			
2	G24_uf (fF, noC)	vs	G24_u (dF, noC)
3	G24_f (fF, fC, OICB)	vs	G24_1 (dF, fC, OICB)
4	G24_f (fF, fC, OICB)	vs	G24_2 (dF, fC, ONICB)
Dynamic objectives: Unconstrained vs Fixed constraints			
5	G24_u (dF, noC)	vs	G24_1 (dF, fC, OICB)
6	G24_2u (dF, noC)	vs	G24_2 (dF, fC, ONICB)
Fixed constraints vs Dynamic constraints			
7	G24_1 (dF, fC, OICB)	vs	G24_4 (dF, dC, OICB)
8	G24_2 (dF, fC, ONICB)	vs	G24_5 (dF, dC, ONICB)
9	G24_f (fF, fC)	vs	G24_7 (fF, dC, NNAO)
10	G24_3f (fF, fC)	vs	G24_3 (fF, dC, NAO)
No constraint vs Dynamic constraints			
11	G24_u (dF, noC)	vs	G24_4 (dF, dC, OICB)
12	G24_2u (dF, noC)	vs	G24_5 (dF, dC, ONICB)
13	G24_uf (fF, noC)	vs	G24_7 (fF, dC)
Moving constraints expose better optima vs not expose optima			
14	G24_3f (fF, fC)	vs	G24_3 (fF, dC, NAO)
15	G24_3 (fF, dC, NAO)	vs	G24_3b (dF, dC, NAO)
Connected feasible regions vs Disconnected feasible regions			
16	G24_6b (1R)	vs	G24_6a (2DR, hard)
17	G24_6b (1R)	vs	G24_6d (2DR, hard)
18	G24_6c (2DR, easy)	vs	G24_6d (2DR, hard)
Optima in constraint boundary vs Optima NOT in constr boundary			
19	G24_1 (dF, fC, OICB)	vs	G24_2 (dF, fC, ONICB)
20	G24_4 (dF, dC, OICB)	vs	G24_5 (dF, dC, ONICB)
21	G24_8b (dF, fC, OICB)	vs	G24_8a (dF, noC, ONISB)

dF	dynamic objective func	fF	fixed objective function
dC	dynamic constraints	fC	fixed constraints
OICB	optima in constraint bound	ONICB	opt. not in constraint bound
OISB	optima in search bound	ONISB	optima not in search bound
NAO	better newly appear optima	NNAO	No better newly appear opt
2DR	2 Disconn. feasible regions	1R	One single feasible region
Easy	easy for mutation to travel between disconn. regions	Hard	less easy to travel among regions
noC	unconstrained problem	SwO	Switched optimum between disconnected regions

5.3.1 Analysing the performance of some dynamic optimisation strategies in solving DCOPs

As discussed in Section 5.1, common DCOPs might have some special characteristics which have not been considered in existing academic research on continuous dynamic optimisation. This raises the question of whether existing dynamic optimisation strategies, which have been designed and tested in unconstrained/domain constrained problems only, can be applied directly to solving DCOPs, and whether the special characteristics of DCOPs might have any effect on the performance of these algorithms. It should be noted that, in order to apply existing dynamic optimisation algorithms directly to solving DCOPs without changing anything, the constraint handling task must be made transparent to the DO algorithms by using methods like penalty functions.

The purpose of this section is to investigate whether the dynamic optimisation strategies commonly used in existing literature can be applied directly to solving DCOPs. I also study whether the special characteristics of DCOPs might have any effect on the performance of these strategies. If there are such effects, we will analyse to see which effects are caused by the combined constraint handling techniques, and which are caused by the nature of the dynamic optimisation strategies regardless of the use of constraint handling. The results of the analysis will also give us insight understanding of how to design suitable algorithms for solving DCOPs.

The strategies that we are going to consider are (1) introducing diversity, (2) maintaining diversity and (3) tracking the previous optima. These three are among the four most commonly used strategies (the other is memory-based strategy) to solve dynamic optimisation problems. The diversity-introducing strategy was proposed based on the assumption that by the time a change happens in the environment, an evolutionary algorithm might have already converged on a specific area and hence would lose its ability to deal with changes in other areas of the landscape. Because of that, it is necessary to increase the diversity level in the population, either by increasing the mutation rate or re-initialising/re-locating the individuals. This strategy has been reviewed in details in Subsection 2.1.2.

The diversity-introducing strategy above requires that changes must be detectable. To avoid this disadvantage, the diversity-maintaining strategy was introduced by which the diversity of the population is always maintained to deal with any possible dynamic without explicitly

detecting changes. This strategy has been reviewed in details in Subsection 2.1.3.

The third strategy, tracking-previous-optima, is found in various approaches reviewed in Section 2.1. It is used in situations where it is assumed that the optima might just slightly change and hence it would be better to focus on observing the nearby places of the current optima to detect changes and "track" the movement of these optima. Similar to the two strategies above, the tracking strategy has also been used since the very early days of dynamic optimisation (Cobb 1990, Vavak *et al.* 1995) and it has always been one of the main strategies for solving DOPs. Recently this strategy has usually been combined with the diversity maintaining/introducing strategy to achieve better performance. Typical examples are the multi-population/multi-swarm approaches (firstly proposed in (Oppacher & Wineberg 1999, Branke *et al.* 2000, Ursem 2000)), where multiple sub-populations are used to maintain diversity and each sub-population/sub-swarm focuses on tracking one singe optimum.

Another strategy that is also commonly used in dynamic optimisation algorithms is the memory-based strategy. In this chapter I do not carry out any experiment directly on the performance of this strategy but leave this task for a future investigation. However, that omission does not mean that we cannot draw any implication about the performance of memory-based in solving DCOPs. The focus of this chapter on the diversity-maintaining/introducing strategies alone would still be beneficial to evaluating how effective a memory-based approach can be in solving a DCOP. This is because, as pointed out by Branke (Branke 2001*b*), in memory-based approaches the memory strategy cannot be used alone but needs to be integrated with some diversity-maintaining/introducing strategies. As a result, if an integrated diversity strategy used in a memory-based algorithm is affected by the characteristics of DCOPs, we can conclude that the memory-based algorithm itself would also be affected.

5.3.2 Chosen algorithms and experimental settings

Chosen algorithms

To evaluate the performance of the three strategies mentioned above in DCOPs, I choose to test two canonical algorithms: *triggered hyper-mutation GA* (HyperM (Cobb 1990)) and *random-immigrant GA* (RIGA (Grefenstette 1992)). HyperM represents the "introducing diversity" and "tracking previous optima" strategies and RIGA represents the "maintaining diversity" strategy. HyperM is basically a basic GA with an adaptive mechanism to switch from a low mutation rate

(*standard-mutation-rate)* to a high mutation rate (*hyper-mutation-rate)* depending on whether there is a degradation of the best solution in the population or not. When there is no degradation of the best solution, the algorithm uses the standard-mutation-rate as basic GA. However, when a drop in value of the best solution (possibly caused by an environmental change) is detected, the algorithm temporarily increases the mutation rate to the high *hyper-mutation-rate* to cope with the change. Because it firstly focuses on observing the current optimum to detect any possible changes and then it increases the level of diversity to "track" the movement when a change is detected, the HyperM algorithm represents the "*tracking-previous-optima strategy*". The algorithm also represents the "*diversity introducing strategy"* because it increases its mutation rate whenever it knows that a change happens.

Different from HyperM, RIGA represents the "*diversity-maintaining*" strategy. This algorithm is also a derivative of basic GA in which in addition to using the standard mutation rate, after the mutation step a fraction of the population is replaced by randomly generated individuals in every generation. That fraction of the population is determined by a *random-immigrant-rate* (also named *replacement rate*). By continuously replacing a part of the population with random solutions, the algorithm is able to maintain diversity throughout the search process to cope with dynamic environments.

There are four reasons for me to choose these two algorithms to test. First, the strategies/mechanisms used in these two algorithms are still commonly used in most current state-of-the-art dynamic optimisation algorithms. As a result, it might be possible to generalise the conclusions we get from testing these two algorithms to many other algorithms. Second, the diversity maintaining/introducing and tracking implementation used in these two algorithms are very simple and straightforward, making it easy to test and analyse the behaviours of the algorithms. Third, because these two algorithms are very well studied, using them in the experiment would facilitate us in comparing new experimental data with existing results. Finally, because both algorithms are developed from basic GA (actually the only difference between HyperM/RIGA and basic GA is the mutation strategy), it would be easier to compare/analyse their performance against each other. Basic GA can also be used as the foundation to develop other strategies to work with DCOPs, then compare their performance with HyperM and RIGA.

To discover if HyperM and RIGA work well on the tested problems, I also compare their performance with basic GA in our experiments. With the slightly higher than normal mutation

rate that I chose (rate=0.15; see next subsection for detail of parameter settings and reasons to choose the settings), to some extent the basic GA and HyperM can also be considered as representatives of the diversity-maintaining strategy. The three algorithms HyperM, RIGA and GA will be evaluated on the G24 benchmark set described in Section 5.2.

In the next subsections readers might notice that in some tables/figures in this section we include not only the three algorithms GA/RIGA/HyperM but also another algorithm which have not been introduced yet. That algorithm will be introduced and analysed in the later sections. For now, in this section we will only focus on the data relating to the elitism and non-elitism versions of GA, RIGA and HyperM.

Another point to be noted in the following experiments is that, although the full standard deviation data is provided in Table 5.7 (page 120), the test results in most of the following graphs are presented with the mean values only. The reason for not presenting the standard deviations in these graphs is due to the technical difficulties in presenting them in the graphs while still maintaining the purpose of allowing readers to compare the performance of all seven algorithms. However, given the data we got, we believe it is sufficient to just compare algorithms using the mean values because in most cases the standard deviations are very small compared to the mean values, and hence should not have any significant impact on deciding the difference between algorithms. In the few cases where an algorithm's error has large standard deviations, its corresponding mean values are also significantly worse than that of other algorithms and hence it is obvious that in these cases the considered algorithm also has worse performance.

Parameter settings

Table 5.6: Test settings for all algorithms used in the paper.

All algorithms (exceptions below)	Pop size	25
	Elitism	Elitism & non-elitism if applicable
	Selection method	Non-linear ranking as in (Michalewicz n.d.)
	Mutation method	Uniform, $P = 0.15$
	Crossover method	Arithmetic, $P = 0.1$
HyperM	Triggered mutation rate	Uniform, $P = 0.5$ as in (Cobb 1990)
RIGA	Random-immigrant rate	$P = 0.3$ as in (Grefenstette 1992)
GA+Repair	Search pop size	20
	Reference pop size	5
	Replacement rate	0 (default is 0.25 as in (Michalewicz n.d.))
Benchmark problem settings	Number of runs	50
	Number of changes	10
	Change frequency	1000 function evaluations
	ObjFunc severity k	0.5 (medium), except G24_6a/b/c/d where $k = 1$ (large severity)
	Constr. severity S	20 (medium)

Table 5.6 (page 113) shows the detailed parameter settings for HyperM, RIGA and GA. To create a fair testing environment, the parameters of all tested algorithms are set to similar values or the best known values if possible. All algorithms use real-valued representation. For the base mutation rate of the algorithms, I use a mutation rate of 0.15, which is the average value of the best mutation rates commonly used (for medium to high severity level of changes) for GA-based algorithms in various existing studies on continuous dynamic optimisation, which are 0.1 (Cobb & Grefenstette 1993, Richter 2009, Richter & Yang 2009), 0.15 (Cobb & Grefenstette 1993) and 0.2 (Branke *et al.* 2000, Branke 1999, Ayvaz *et al.* 2006). For HyperM and RIGA, I use the best *hyper-mutation-rate* and *random-immigrant-rate* parameter values observed in the original papers (Cobb 1990) (Grefenstette 1992) for this experiment. I also use the same implementations as described in (Cobb 1990) and (Grefenstette 1992) to reproduce these two algorithms. The crossover rate of 0.1 is chosen for all algorithms because according to our analysis this is one of the few settings where all tested algorithms perform well in the G24 benchmark set (see Chapter 6 for more details). All algorithms have a population size of 25. This population size is chosen based on the hardness level of the tested problems. The population size of 25 would also facilitates us in comparing the algorithms with some existing constraint-handling algorithms which also have the default population size of 25.

The algorithms were tested in 18 benchmark problems described in section 5.2 in two levels of change severity: medium and high except in G24_6a/b/c/d where the severity is always high (high severity is a property of these four problems). Because the observed behaviours of the tested algorithms are the same for both cases of severity, in this chapter I will only present results for the medium severity case.

It might be interesting to investigate if the default/best parameter values from previous literature are also the most suitable parameter values for solving the problems in this benchmark set. Because of that, in addition to the experiments in this chapter I also carry out a further study of the effect of different parameter values of the base mutation rates, hyper-mutation rates, random-immigrant rates and crossover rates on algorithm performance. The experimental results and discussion for this analysis can be found in Section 6.4.

Constraint handling

To apply existing dynamic optimisation algorithms directly to solving DCOPs, we also need to integrate them with a constraint handling mechanism. That constraint handling mechanism should not interfere with or change the original dynamic optimisation strategies in any way so that we can correctly evaluate whether the original dynamic optimisation strategies would still be effective in solving DCOPs. To satisfy that requirement, I chose to use the penalty function approach because it is the simplest and easiest way to apply existing unconstrained dynamic optimisation algorithms directly to solving DCOPs without changing anything in the algorithms. In this chapter I present the test results using the penalty function proposed in (Morales & Quezada 1998). I chose this penalty function because it is reportedly effective in solving difficult numerical problems and more importantly because it does not require users to choose any penalty factor or other parameter. This allows us to apply existing dynamic optimisation algorithms directly to solving DCOPs without any additional effort. There are more sophisticated and better penalty methods in the literature, but because such methods might require additional tasks to choose the appropriate penalty factors/parameters or to choose the appropriate dynamic penalty techniques, they might prevent algorithm users from applying existing DO strategies directly and quickly to solving DCOPs. I also tested the algorithms in cases where the infeasible solutions are penalised with various penalty values which are specifically chosen so that the fitness values of infeasible solutions are always worse than or equal to that of feasible solutions.

The experimental results in these cases, however, are not shown because there is no significant difference in the test results compared to the case of using the penalty function proposed in (Morales & Quezada 1998).

Performance measures

To measure the performance of the algorithms in this particular experiment, I firstly modify an existing measure: the *modified offline error* proposed in (Branke & Schmeck 2003). The modified offline error is measured as the average over, at every evaluation, the error of the best solution found since the last change of the environment. This measure is always greater than or equal to zero and would be equal to zero for a perfect performance.

Because the measure above is designed for unconstrained environments, we need to modify it to evaluate algorithm performance in constrained environments. This is because in constrained environments we are interested in evaluating the ability of algorithms in finding not every good solutions but only *good feasible solutions*. The modification is simple. At every generation, instead of considering the best errors/fitness values of *any* solutions regardless of feasibility as implemented in the original measure, in my modification I only consider the best fitness values / best errors of *feasible* solutions at each generation. The fitness and errors of infeasible solutions will not be counted, regardless of their values. If in any generation there is no feasible solution, the measure will take the *worst possible value* that a feasible solution can have for that particular generation. The formula of the modification for the *offline error* measure is given in equation (5.3). We call this measure *modified offline error for DCOPs*, or *offline error* for short.

$$E_{MO} = \frac{1}{n} \sum_{j=1}^{n} e_{MO}(j) \tag{5.3}$$

where n is the number of generations so far, and $e_{MO}(j)$ is the best *feasible* error since the last change gained by the algorithm at the generation j.

The measure that I have modified above is useful in evaluating the overall performance of the tested algorithms to see if they work well in the tested problems. However, it does not provide us with enough detailed information to analyse why a particular algorithm works well or not well in a particular problem. This gap motivates me to propose some new performance measures to assist algorithm designers in analysing the behaviours of dynamic optimisation algorithms. Among these new measures, five will be introduced in this section and other two

algorithm-specific measures will be introduced in Section 5.4.

For this section, I propose five new measures. The first two measures are the *recovery rate* (RR) and the *absolute recovery rate* (ARR) to analyse the convergence behaviour of algorithms in dynamic environments. The *recovery rate* (RR) measure is used to analyse *how quick it is for an algorithm to recover from a performance drop when a change happens and to start converging to a new solution before the next change happens.* The new solution is not necessarily the global optimum.

$$RR = \frac{1}{m}\sum_{i=1}^{m}\frac{\sum_{j=1}^{p(i)}\left[f_{best}\left(i,j\right)-f_{best}\left(i,1\right)\right]}{p\left(i\right)\left[f_{best}\left(i,p\left(i\right)\right)-f_{best}\left(i,1\right)\right]} \tag{5.4}$$

where $f_{best}\left(i,j\right)$ is the fitness value of the best feasible solution since the last change found by the tested algorithm until the jth generation of the change period i , m is the number of changes and $p\left(i\right), i = 1 : m$ is the number of generations at each change period i. The RR score would be equal to 1 in the best case where the algorithm is able to recover and converge to a solution immediately after a change, and would be equal to zero in case the algorithm is unable to recover from the drop at all.

The RR measure only tells us if the considered algorithm converges to a solution and if it converges quickly. It does not indicate whether the converged solution is the global optimum. For example, RR can still be equal to 1 if the algorithm does nothing but keep re-evaluating the same solution. Because of that, we need another measure: the *absolute recovery rate* (ARR). This measure is very similar to the RR but is used to analyse *how quick it is for an algorithm to start converging to the global optimum before the next change happens*:

$$ARR = \frac{1}{m}\sum_{i=1}^{m}\frac{\sum_{j=1}^{p(i)}\left[f_{best}\left(i,j\right)-f_{best}\left(i,1\right)\right]}{p\left(i\right)\left[f^{*}\left(i\right)-f_{best}\left(i,1\right)\right]} \tag{5.5}$$

where $f_{best}\left(i,j\right)$ is the best solution since the last change found by the tested algorithm until the jth generation of the change period i , $f^{*}\left(i\right)$ is the global optimal value of the landscape at the ith change, m is the number of changes and $p\left(i\right), i = 1 : m$ is the number of generations at each change period i. The ARR score would be equal to 1 in the best case when the algorithm is able to recover and converge to the global optimum immediately after a change, and would be equal to zero in case the algorithm is unable to recover from the drop at all. It should be noted that the score of ARR should always be less than or equal to that of RR. In the ideal case

(converged to global optimum), ARR should be equal to RR.

The RR and ARR measures can be used together to indicate if an algorithm is able to converge to the global optimum within the given time frame between changes and if so how fast it takes to converge. The combination can also be used to analyse if the cause for an algorithm to work not well is slow convergence or pre-mature convergence. The *RR-ARR diagram* in Figure 5.2 (page 118) shows some guidelines to analyse the behaviours of tested algorithms given the scores of ARR and RR. In this diagram the RR and ARR scores can be represented as the x and y coordinations of a point, which always lies on the diagonal thick line or inside the shaded area.

By looking at the position of the point, we will be able to analyse the behaviour of the corresponding algorithm. First, if the point lies on the thick diagonal line (where $RR = ARR$) like point A, we can conclude that the algorithm A has been able to recover from the change and converged to the new global optimum. Along that line, the closer the point is to the right, the faster the algorithm was in recovering and re-converging, and vice versa. Second, if the point lies inside the shaded area (e.g. point B, C, D), the algorithm either has converged to a local solution or has not been converged yet. In addition, the closer the point is to the optimum line, the closer the algorithm is to the global optimum. Third, points in the top right corner of the shaded area (like point B) show that the algorithm has been able to recover fast and was able to achieve a good performance (although not yet found the global optimum). The closer the point is to the top right corner, the better the performance and the faster the recovery speed. Fourth, points in the bottom-right corner (like point C) shows that the algorithm has been likely converged to a local solution. In this case the algorithm recover fast but then was trapped in a local solution far from the global optimum. The closer a point is to the bottom-right corner, the more likely that the algorithm is trapped. Fifth, points near the bottom-left corner (like point D) show that the algorithm has recovered slowly and has likely not converged yet.

We can also use the diagram to compare and analyse the behaviour of different algorithms. For example in this figure we can see that algorithm A found the best solution after change, following by algorithms B, D, C while algorithm B was able to recover and converge fastest, following by algorithm C, A, and D. In case we need the best solution after change with no limit in time, we can choose algorithm A over B. However, if we need a good solution quickly, then algorithm B might be more preferrable.

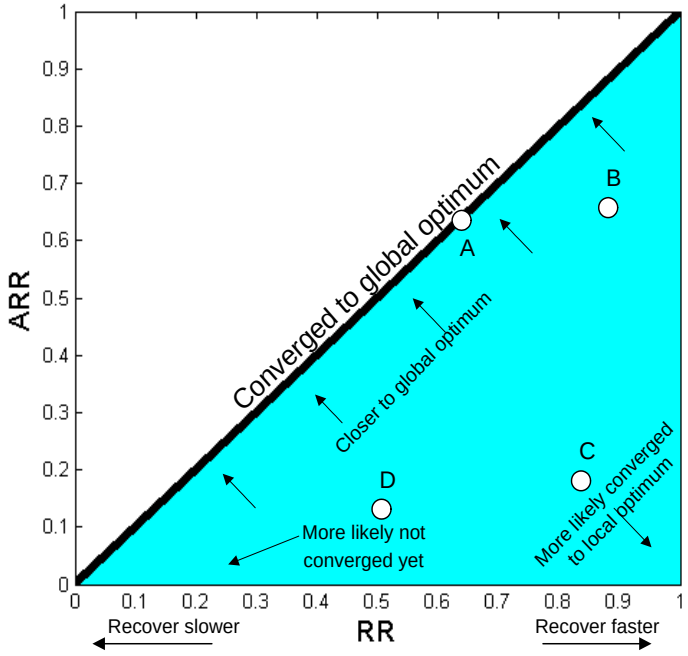

Figure 5.2: The RR-ARR diagram to analyse the convergence behaviour/recovery speed of an algorithm given its RR and ARR scores. In this diagram the RR and ARR scores can be represented as the x and y coordinations of a point, which always lies on the diagonal thick line or inside the shaded area.

It should be noted however that in order to use the measure ARR we need to know the global optimum value at each change period.

To analyse the ability to balance feasibility/infeasibility of algorithms using the diversity maintaining/introducing strategies as RIGA/HyperM in DCOPs, I propose a third measure: *percentage of selected infeasible individuals*. Among the individuals selected for the next generation, this measure counts the percentage of those that are infeasible. The average score of this measure (over all tested generations) is then compared with the percentage of infeasible areas over the total search area of the landscape. If the considered algorithm is able to treat *infeasible* diversified individuals and *feasible* diversified individuals on an equal basis (and hence to maintain diversity effectively), the two percentage values should be equal.

To analyse the behaviour of algorithms using triggered-mutation mechanism as HyperM, I also propose a fourth measure: *triggered-time count*, which counts the number of times the hyper-mutation-rate is triggered by the algorithm, and a fifth measure: *detected-change count*, which counts the number of triggers actually associated with a change. For HyperM, triggers

associated with a change are those that are invoked by the algorithm within ν generations after a change, with ν is the maximum number of generations (five in our implementation) needed for HyperM to detect a drop in performance. These two measures indicate how many times an algorithm triggers its hyper-mutation; whether each trigger time corresponds to a new change; and if there is any change goes undetected during the search process.

It should also be noted that all the measures used in this chapter are designed specifically for dynamic problems. This creates an issue for our experiments because in the G24 benchmark set there are not only dynamic problems, but also stationary problems. To overcome this issue and to create a fair, normalised testing environment, in the experiments in this chapter we consider stationary problems a special type of dynamic problem which still have "changes" after each 1000 function evaluations as other dynamic problems. However, in stationary problems the "changes" do not alter the search landscape. That way we can apply the dynamic optimisation measures to both stationary and dynamic problems and can compare the performance of algorithms fairly in both types of problems.

5.3.3 Experimental results and analyses

The full results of the tested algorithms in all 18 benchmark problems are presented in Table 5.7 (page 120). The data in this table is provided for reference purpose only because to achieve a better understanding of how existing dynamic optimisation strategies work in DCOPs and how each characteristic of DCOPs would affect the performance of existing dynamic optimisation algorithms, we further analyse the results by studying them from different perspectives. First, we summarise the average performance of the tested algorithms in each major group of problems (see test results in Figure 5.3, page 121) to have an overall picture of the behaviours of each algorithm in different types of problems. Second, we investigate the effect of each problem characteristic on each algorithm by analysing their performance in 21 test cases (pair of almost identical problems, one with a particular characteristic and one without) as shown in Table 5.5 of Section 5.2 (see test results in Figure 5.4, page 122 and Figure 5.5, page 123). For each particular algorithm, I also carry out some further analyses using the five newly proposed measures mentioned above. Details of these analyses will be described in the next subsections.

Table 5.7: Averaged modified offline errors of all tested algorithms in all 18 problems after 50 runs.

	G24-u (dF,noC)		G24-1 (dF, fC)		G24-f (fF, fC)	
Algorithm	mean	stdDev	mean	stdDev	mean	stdDev
.GA-noElit	0.298	0.051	0.609	0.064	0.676	0.085
.RIGA-noElit	0.221	0.025	0.493	0.045	0.546	0.072
.HyperM-noElit	0.206	0.035	0.361	0.065	0.226	0.056
.GA-elit	**0.106**	0.035	0.459	0.057	0.154	0.083
.RIGA-elit	0.149	0.025	0.346	0.046	0.178	0.051
.HyperM-elit	0.111	0.026	0.384	0.065	0.149	0.053
.GA+Repair	0.468	0.059	**0.226**	0.024	**0.041**	0.011

	G24-uf (fF, noC)		G24-2 (dF, fC)		G24-2u (dF,noC)	
Algorithm	mean	stdDev	mean	stdDev	mean	stdDev
.GA-noElit	0.464	0.064	0.356	0.049	0.159	0.041
.RIGA-noElit	0.342	0.032	0.264	0.035	0.107	0.019
.HyperM-noElit	0.124	0.041	0.257	0.045	0.130	0.022
.GA-elit	0.063	0.022	0.288	0.050	0.073	0.017
.RIGA-elit	0.069	0.020	**0.246**	0.037	0.091	0.024
.HyperM-elit	**0.053**	0.012	0.253	0.043	**0.068**	0.016
.GA+Repair	0.218	0.018	0.281	0.036	0.294	0.029

	G24-3 (fF,dC)		G24-3b (dF,dC)		G24-3f (fF, fC)	
Algorithm	mean	stdDev	mean	stdDev	mean	stdDev
.GA-noElit	0.760	0.099	0.657	0.097	0.886	0.179
.RIGA-noElit	0.538	0.047	0.500	0.038	0.651	0.055
.HyperM-noElit	0.411	0.052	0.459	0.069	0.256	0.057
.GA-elit	0.289	0.049	0.457	0.084	0.158	0.058
.RIGA-elit	0.308	0.048	0.386	0.051	0.167	0.048
.HyperM-elit	0.243	0.050	0.394	0.088	0.128	0.051
.GA+Repair	**0.156**	0.008	**0.171**	0.019	**0.025**	0.008

	G24-4 (dF, dC)		G24-5 (dF,dC)		G24-6a(dF 2DR,hard)	
Algorithm	mean	stdDev	mean	stdDev	mean	stdDev
.GA-noElit	0.621	0.101	0.379	0.067	0.529	0.108
.RIGA-noElit	0.490	0.053	0.293	0.046	0.366	0.030
.HyperM-noElit	0.469	0.057	0.275	0.034	0.383	0.051
.GA-elit	0.453	0.075	0.266	0.029	0.674	0.157
.RIGA-elit	0.421	0.047	0.240	0.035	**0.333**	0.050
.HyperM-elit	0.426	0.075	0.248	0.039	0.491	0.071
.GA+Repair	**0.211**	0.015	**0.236**	0.024	0.431	0.074

	G24-6b (dF,fC 1R)		G24-6c(dF 2DR,easy)		G24-6d(dF 2DR,hard)	
Algorithm	mean	stdDev	mean	stdDev	mean	stdDev
.GA-noElit	0.448	0.054	0.446	0.041	0.543	0.127
.RIGA-noElit	0.331	0.035	0.329	0.039	0.366	0.040
.HyperM-noElit	0.340	0.046	**0.323**	0.037	0.370	0.046
.GA-elit	0.408	0.057	0.441	0.052	0.510	0.075
.RIGA-elit	**0.309**	0.039	0.325	0.029	**0.342**	0.057
.HyperM-elit	0.390	0.039	0.394	0.051	0.456	0.041
.GA+Repair	0.427	0.048	0.390	0.038	0.354	0.038

	G24-7 (fF, dC)		G24-8a(dFnC,ONISB)		G24-8b (dFfC,OICB)	
Algorithm	mean	stdDev	mean	stdDev	mean	stdDev
.GA-noElit	0.721	0.088	0.426	0.050	0.835	0.068
.RIGA-noElit	0.543	0.059	0.346	0.031	0.719	0.071
.HyperM-noElit	0.495	0.053	0.374	0.043	0.681	0.072
.GA-elit	0.316	0.053	**0.266**	0.028	0.662	0.056
.RIGA-elit	0.416	0.068	0.304	0.028	0.598	0.064
.HyperM-elit	0.315	0.062	0.279	0.028	0.608	0.071
.GA+Repair	**0.181**	0.017	0.300	0.033	**0.251**	0.051

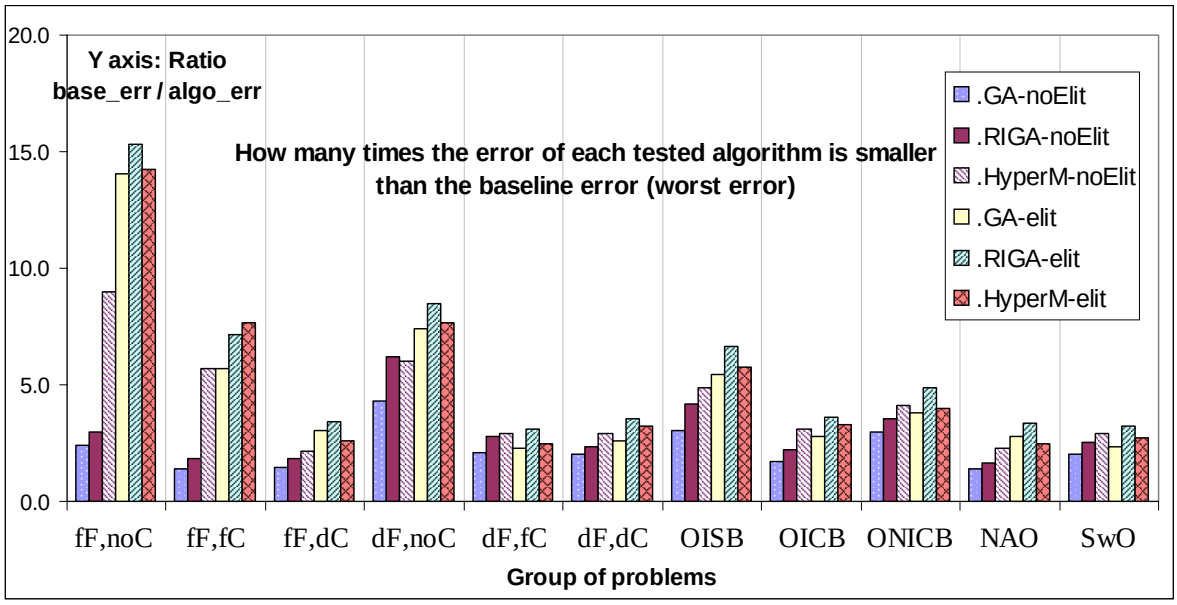

Figure 5.3: This figure shows the performance of the elitism (-elit) and non-elitism (-noElit)versions of existing dynamic optimisation algorithms (GA, RIGA and hyperM) in different groups of problems. This figure shows us not only how each algorithm perform in each particular group of problems, but also in which group do the algorithms perform better or worse. Algorithms' performance is evaluated based on their modified offline error (or "error" in short) as follows. First, the worst (largest) error among all algorithms is recorded as the base line error. Then we calculate the ratio between the base line error and the error of each algorithm in each problem to see how many times their performance is better (smaller) than the base line error. This ratio is represented in the vertical axis. The horizontal axis shows different group of problems. Explanations for the abbreviations in this figure and all other figures in this paper are as follow: noC: No Constraint; fC: fixed Constraint; fF: fixed objective Function; dC: dynamic Constraint; dF: dynamic objective Function. O(N)ICB: Optimum (Not) In Constraint Boundary; (N)NAO: (No) Newly Appearing Optimum; ONISB: Optimum (Not) In Search Boundary; 1R: One single feasible region; 2DR: Two Disconnected feasible Regions; easy/hard: Easy/difficult path between disconnected regions; SwO: Switched global Optimum between disconnected regions.

The experiments and analysis results show some interesting and in some cases even surprising or counter-intuitive findings, which will be shown in the following subsections.

The effect of elitism on algorithm performance

The summarised results in groups of problems (Figure 5.3) and the pair-wise comparisons in Figure 5.4 (page 122) and Figure 5.5 (page 123) reveal an interesting effect of elitism on both unconstrained and constrained dynamic cases in our test. This is the fact that, the elitism versions of GA/RIGA/HyperM perform better than their non-elitism counterparts in most tested problems. The reason for this effect (with evidence shown in the next paragraph) is that elitism

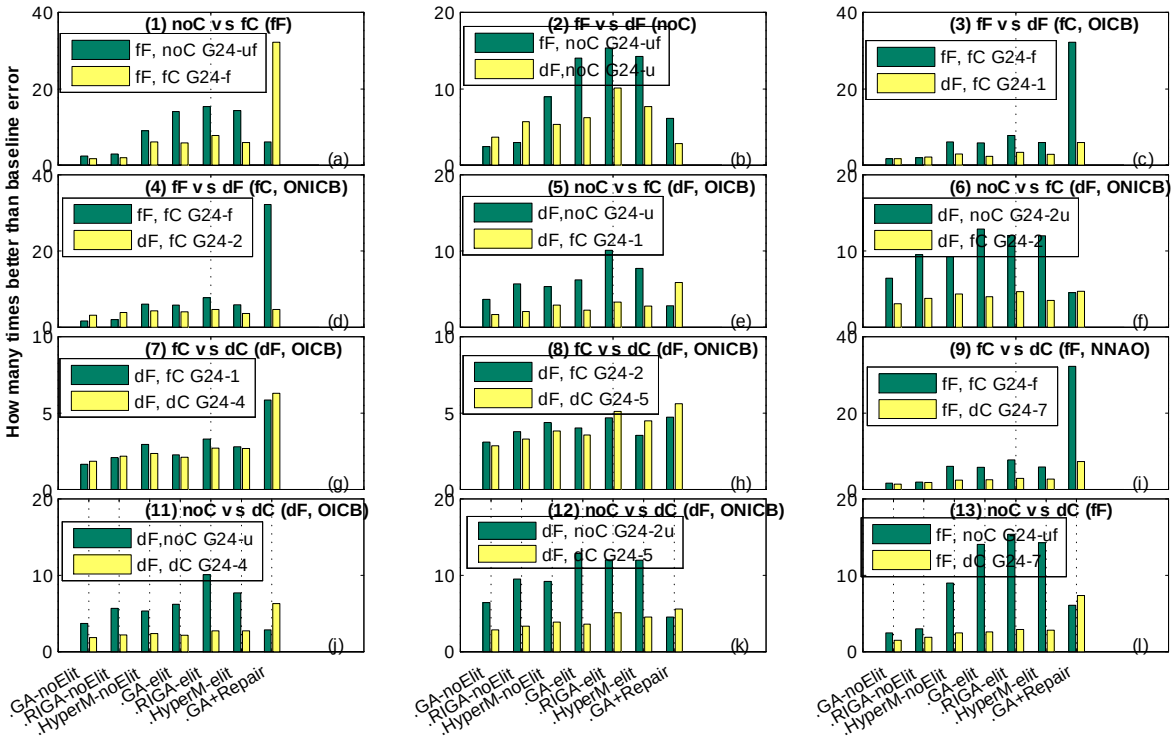

Figure 5.4: This figure summarises the effect of twelve different problem characteristics on the performance of the non-elitism and elitism versions of GA, RIGA, and HyperM, and the repair-based algorithm GA+Repair (to be introduced later in Subsection 5.4.4). Each subplot represents algorithm performance in a pair of almost identical problems (one has a special characteristic and the other does not). The heights of the bars in each subplot indicate how well the tested algorithms perform in solving the pair of problems. The higher the bars, the better the performance. Each pair of adjacent bars represent the performance of one algorithm in a pair of problems. The larger the difference between the bar heights, the larger the difference in performance, and hence the greater the impact of the corresponding DCOP characteristic on algorithm performance. It should be noted that pair (10) is not included in this figure because it is identical to pair (14) in Figure 5.5. Algorithms' performance is evaluated based on the ratio between the base line error (as described in the caption of Figure 5.3) and the error of each algorithm. This is to see how many times their performance is better (smaller) than the base line error. This ratio is represented in the vertical axis. The title of each subplot represents the test case number (in brackets) followed by an abbreviated description of the test case. Explanations for the abbreviations can be found in the caption of Figure 5.3.

helps algorithms with diversity-maintaining strategies to converge faster. This effect is caused by the nature of the dynamic optimisation strategies, i.e. is independent of the combined constraint handling techniques.

It should be noted that our detailed analysis (not shown) also pointed out that for HyperM, elitism only has positive effect in case the base mutation rate of HyperM is large enough (0.15 or larger), i.e. only in cases where the diversity-introducing strategy (high base mutation rate)

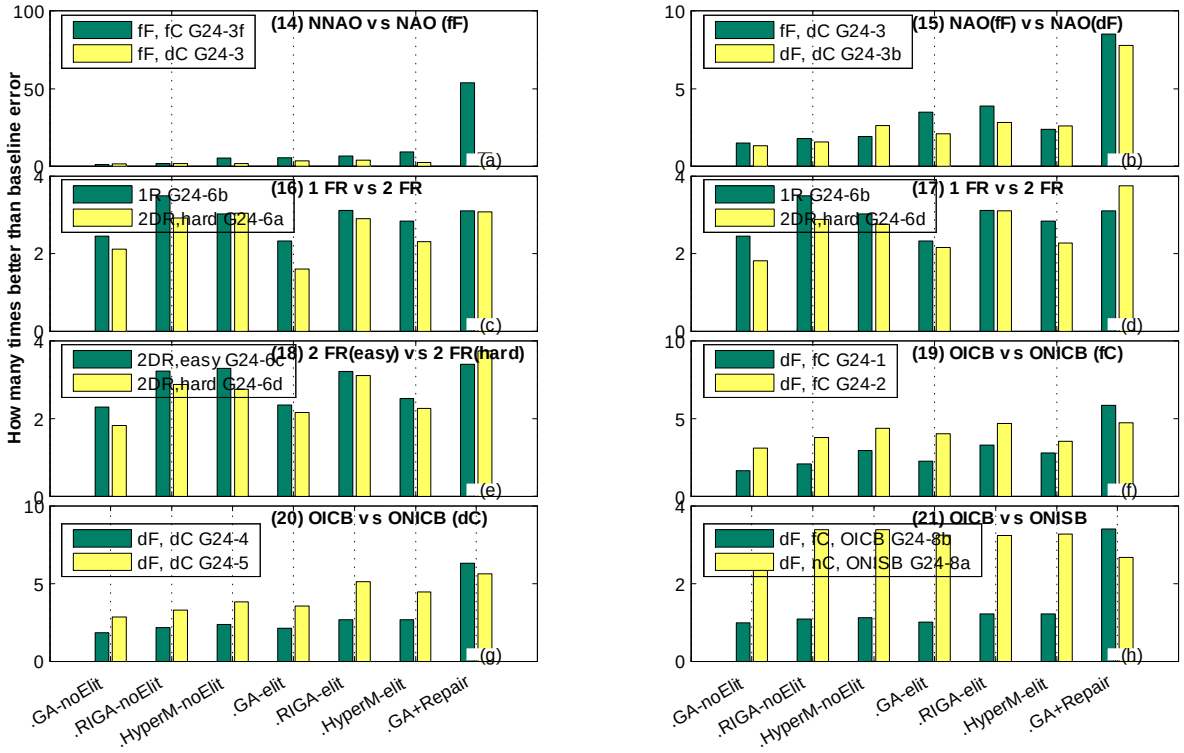

Figure 5.5: This figure summarises the effect of the other eight different problem properties on the performance of GA, RIGA, HyperM (elitism and non-elitism versions) and GA+Repair. Instruction to read this figure can be found in the caption of Figure 5.4.

is combined with the diversity-maintaining strategy. In cases where the base mutation rate of HyperM is lower (smaller than 0.15), elitism actually has a negative effect and makes the algorithm become more prone to pre-mature convergence. This negative effect of HyperM is also caused by the nature of this strategy.

To study the reasons for the inefficiency of GA/RIGA/HyperM in the non-elitism case compared to the elitism case, I use the two measures proposed in Subsection 5.3.2: *recovery rate* (RR) and *absolute recovery rate* (ARR). The scores of the tested algorithms on these measures are shown and analysed in Figure 5.6 (page 125). Based on the guidelines in Figure 5.2 (page 118), we can use the coordinations of the RR/ARR scores in the diagram to analyse the convergence behaviour of the algorithms as well as the reasons behind the impact of elitism on the performance of GA/RIGA and HyperM. First, the diagram shows us that none of the algorithms are close to the optimum line, meaning that overally there are problems/ change periods where the algorithms have not been able to converge to the global optimum. Second, for GA/RIGA, we

can see that the elitism versions of the tested algorithms are closer to the top-right corner while their non-elitism are closer to the bottom-left corner. Because in the *RR-ARR diagram* the top-right corner represents faster/more accurate recovery/convergence and the bottom-left corner represents the reverse thing, it means that non-elitism makes GA/RIGA converge *slower/less accurately*. The diagram also shows that RIGA-elit has the best overall recovery speed an convergence accuracy, following by RIGA-noElit, GA-elit and HyperM-elit, all three have almost the same RR/ARR scores. Third, for HyperM, we see that its elitism version is also closer to the top-right corner while its non-elitism version is closer to the bottom-right corner. Because the bottom-right corner represents faster recovery but more liley to converge to local solutions, the result here suggests that the non-elitism version of HyperM is more suceptible to premature convergence.

The results hence show that the high diversity maintained by the random-immigrant rate in RIGA and the high base mutation rate in GA and HyperM comes with a trade-off: the convergence speed is affected. In such situation, elitism can be used to speed up the convergence process. Elite members can guide the population to exploit the good regions faster while still maintaining diversity.

Effect of infeasible areas on maintaining/introducing diversity

Another interesting observation from our experiments is that the presence of constraints makes the performance of diversity-maintaining/introducing strategies less effective when they are used in combination with the tested penalty functions. This behaviour can be seen in Figure 5.3 where the performance of all algorithms in the unconstrained dynamic case (dF+noC) is significantly better than their performance in all dynamic constrained cases (dF+fC, fF+dC, dF+dC). This behaviour can also be seen in the more accurate comparisons from the pair-wise comparisons in Figure 5.4 (page 122) and Figure 5.5 (page 123), for each pair of problems in which one has constraints and the other does not, GA, RIGA and HyperM always perform worse in the problem with constraints (see subplots a, e, f, j, k, l in Figure 5.4 and subplot h in Figure 5.5).

The reason for this inefficiency of diversity-maintaining/introducing strategies in solving problems with constraints is that the use of the tested penalty functions prevents the diversity-maintaining/introducing mechanisms from working effectively. In solving unconstrained dynamic problems, all diversified individuals generated by the diversity maintaining strategy or

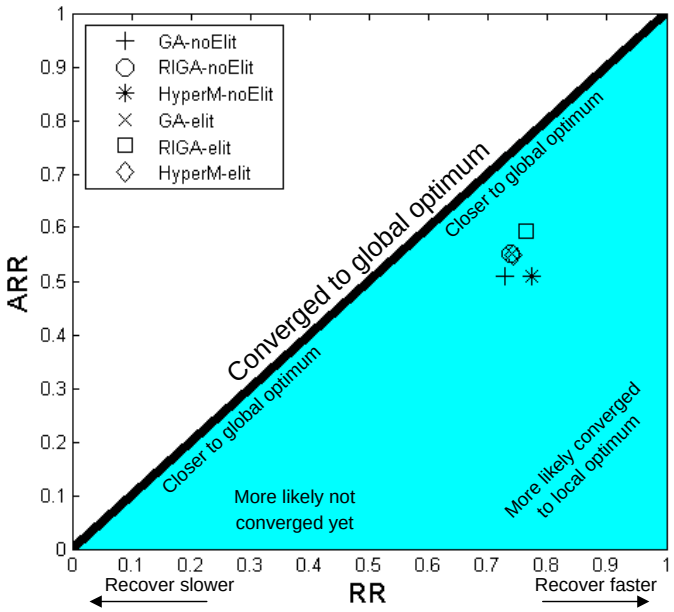

Figure 5.6: This figure shows the mapping of the RR/ARR scores of GA, RIGA, and HyperM to the RR-ARR diagram. The scores are averaged from the results of the above algorithms on all 18 tested problems. Both elitism and non-elitism versions of these algorithms are tested.

the diversity introducing strategy are useful because they contribute to either (1) detecting new appearing optima or (2) finding the new place of the moving optima. In DCOPs, however, I found that only the diversified individuals that are feasible can become fully beneficial to the combination of GA/RIGA/HyperM and penalty methods. The reasons for this behaviour are explained below.

For infeasible diversified individuals, there are two difficulties that prevent them from being useful in existing dynamic optimisation strategies. First, many diversified but infeasible individuals might not be selected for the next generation population because they are penalised with lower fitness values by the penalty functions. Without being selected for the next generation, diversified individuals will not be able to meet the purpose of maintaining/introducing diversity unless they are re-introduced again in the next generation. To demonstrate this drawback, I use the measure *percentage of selected infeasible individuals* proposed in subsection 5.3.2. This measure is used to analyse the relationship between the percentage of infeasible areas over the total search area and the actual percentage of infeasible solutions over the total number of solutions selected for the next generation. If an algorithm is able to treat *infeasible* diversified individuals

and *feasible* diversified individuals on an equal basis (and hence to maintain diversity effectively), the latter percentage should be close to the former percentage. However, as can be seen in table 5.8 (page 126), we can see that in the elitism case the percentage of infeasible solutions in the population (12.5 - 26.3%) is much smaller than the percentage of infeasible areas over the total landscape (60.8%). It means that only a few of diversified, infeasible solutions are retained and hence the algorithms are not able to maintain diversity in the infeasible regions. As a result, if after a change, a new feasible region occurs inside or near the infeasible regions, the algorithms might not be able to react effectively unless diversified individuals are re-introduced at every generation as in the case of RIGA. In the non-elitism case, the percentage of selected infeasible individuals is better than in the elitism case, meaning that these algorithms are able to retain more infeasible individuals, of which some might be diversified solutions. However, in the non-elitism case this higher percentage of infeasible individuals comes with a trade-off of slower/less accurate convergence as shown in the previous subsection 5.3.3. As shown in subsection 5.3.3, this slow convergence leads to the generally poorer performance of the non-elitism algorithms in the test.

Table 5.8: This table shows the average *percentage of selected infeasible individuals* over all 18 problems for each tested algorithm (see Subsection 5.3.2 for descriptions). The last row shows the average *percentage of infeasible areas* over all 18 problems. If an algorithm is able to maintain diversity effectively, its average *percentage of selected infeasible individuals* score should be close to the average *percentage of infeasible areas*.

Algorithms	Percent of infeasible solutions
.GA-elit	12.5%
.RIGA-elit	26.3%
.HyperM-elit	14.8%
.GA-noElit	41.8%
.RIGA-noElit	46.8%
.HyperM-noElit	42.8%
Percentage of infeasible areas	**60.8%**

Second, even if a diversified but infeasible individual is accepted for the next generation, it might still not be able to contribute to the two purposes they are designed for: detecting changes and tracking changes. This inefficiency is also due to the fact that infeasible individuals no longer have their actual fitness value but only penalised fitness values. These penalised fitness values might not accurately reflect the dynamic from environments and hence might not help in detecting and tracking changes. In other words, in the tested dynamic optimisation algorithms

more diversity does not necessarily mean more adaptability.

Effect of switching global optima (between disconnected feasible regions) on strategies that use penalty functions/values

The third finding that I observe from our experiments is the inefficiency of existing dynamic optimisation methods when they are used in combination with the tested penalty functions to solve a special class of DCOPs. This is the class of problems with disconnected feasible regions where the global optimum switches from one region to another whenever a change happens (this is one of the common characteristics of DCOPs as already discussed in section 5.1). In addition, the more separated the disconnected regions are, the more difficult it is for algorithms using penalty functions to solve.

The reason for this behaviour is as follows. In problems with disconnected feasible regions, in order to track the moving optimum from one region to another, it is necessary to have a path going through the infeasible areas that separate the disconnected regions. This path might not be available if we use penalty functions because penalties make it unlikely that infeasible individuals are accepted. Obviously the larger the infeasible areas between disconnected regions, the harder it is to establish the path using penalty methods.

To verify the statement above, I use three test cases (pairs of almost identical problems) provided in Table 5.5 (page 108). They are test cases 16, 17, and 18. In all three test cases the objective functions are the same and the global optimum keeps switching between two feasible regions whenever a change happens. However, the infeasible areas in the problems of each test case are different and hence each test case represents a different dynamic situation. Test case 16 tests the situation where in one problem of the pair (G24_6b) there is a feasible path connecting the two regions and in the other problem (G24_6a) the path now is infeasible, i.e. there is an infeasible area separating two feasible regions. Except for this detail the two problems of the pair are identical. If the use of the tested penalty functions really prevents an algorithm from travelling through the infeasible area, the performance of that algorithm will become worse when the path is infeasible. Test case 17 is the same as test case 16 except that the infeasible area separating two feasible regions has a different shape. Test case 18 tests a different situation where two problems of the pair are almost identical except that in one problem (G24_6c) the infeasible area separating the two feasible regions is small whereas in the other

problem (G24_6d) that infeasible area is large. Again, if the use of the tested penalty functions really prevents an algorithm from searching through infeasible areas, the performance of that algorithm will become worse in the case the infeasible area is larger.

The experimental results in these three test cases (subplot c, d, e in Figure 5.5, page 123) confirm that the hypotheses stated in the beginning of this subsection are true. In subplot c and d, the tested algorithms do suffer when the path between the two regions is infeasible. In subplot e, the larger the infeasible area separating the two regions, the worse the performance of the tested algorithms. All in all, the results prove that the combination of existing dynamic optimisation strategies with the tested penalty functions might be less effective in problems with disconnected regions and switching optima.

Effect of moving infeasible areas on strategies that track the previous global optimum

The fourth interesting finding is the fact that, algorithms that rely on tracking previous global optimum as HyperM might become less effective when solving DCOPs where the moving constraints expose new, better optima without changing the existing optima. The reason is that they might not be able to detect changes in such type of DCOPs. As shown below, this behaviour of tracking-previous-optimum algorithms also leads to an interesting counter-intuitive fact: for this type of algorithm, some DCOPs with dynamic objective functions might become easier to solve than some DCOPs with fixed objective functions. Similar to the case of elitism, the effect of moving infeasible areas on tracking-previous optimum strategies is also caused by the nature of the strategies, i.e. it is independent of the combined constraint handling techniques.

Evidence of this behaviour can be found when we tested the algorithms in the test case 15 (Table 5.5), which is a pair of problems with newly better optima exposed by moving constraints. The two problems in the pair, G24_3 and G24_3b, are almost identical except for that the former has a fixed objective function while the latter has a dynamic objective function which changes whenever the environment changes. In other words, the only difference between the two problems is that in G24_3 the existing optima remain intact after each change while in G24_3b these existing optima change due to the dynamic of the objective function. This characteristic makes G24_3 supposedly *more easier* to solve than G23_3b because it has *fewer* dynamic elements.

However, experimental results in subplot b, Figure 5.5 and in Table 5.7 show that this is only true with RIGA and GA. In the case of HyperM, the fixed objective function in G24_3 actually makes the problem *more difficult* for HyperM to solve than in G24_3b (in subplot b, Figure 5.5 we can see that HyperM's bar in G24_3b is higher than HyperM's bar in G24_3). The results show that the "stationary" of existing optima, which is the only difference between the two problems, must be the reason for the decrease in performance of HyperM in G24_3. This is an interesting example of how stationary can make the problems more difficult, or in other words how dynamic can help to make the problems easier to solve for certain types of algorithms.

Because the only difference between HyperM and RIGA/GA is its triggered-mutation strategy, which is specifically for tracking the existing optimum, the decrease in performance of HyperM must be due to its triggered-mutation strategy. To investigate why HyperM suffers in problems like G24_3 and why the dynamic of existing optima in G24_3b can help the algorithm to improve its performance, the newly proposed measures *triggered-time count* and *detected-change count* (see subsection 5.3.2) are used to analyse how the triggered-hypermutation mechanism works in these two problems. The analysis results indicate that the reason for the less efficiency of HyperM in G24_3 is that the algorithm is unable to detect changes. The algorithm is unable to detect changes because its tracking optima strategy only focuses on monitoring existing optima, and is hence unable to recognise the newly, better optima exposed by the moving constraints. As can be seen in table 5.9 (page 131), the algorithm HyperM either is not able to trigger its hyper-mutation rate to deal with changes (elitism case, *triggered-time count*=0 and *detected-change count*=0) or is not able to trigger its hyper-mutation rate correctly when a change happens (non-elitism case, *triggered-time count*$\sim$164 and *detected-change count*$\sim$1.8). It is interesting to observe that in the non-elitism case, the averaged number of trigger times is relatively high (164.2) but almost none of these trigger times correlates to the changes in the landscape, i.e. almost no change is detected. Instead, these trigger times are caused by the selection process due to the fact that in non-elitism selection the best solution in the population is not always selected for the next generation.

On the contrary, in problems with dynamic objective function like G24_3b, the analysis results indicate that the reason why HyperM works well in this problem is that whenever a change happens, the value of existing optima changes accordingly, hence prompting the algorithm to

trigger its hyper-mutation rate accurately. As can be seen in table 5.9, after eleven changes the number of triggered times correlated to a change is also eleven in both the elitism and non-elitism cases.

Our detailed analysis (not shown) also indicates that the difference in performance of HyperM between G24_3b and G24_3 becomes larger when the base mutation rate becomes smaller. This is because, due to its inability to detect changes in solving problems like G24_3, HyperM is not able to use its triggered hyper-mutation rate. In such case the base mutation rate is the only option for HyperM to track the newly appearing optimum, and the smaller the base mutation rate, the less likely that the algorithm is able to track that moving optimum.

All in all, the test results confirm that algorithms relying on tracking the existing optima as HyperM might become less effective for solving DCOPs where the moving constraints expose new, better optima without changing the existing optima. This is due to the fact that the algorithms might not be able to detect changes.

In addition, the results show that for algorithms like HyperM, in certain cases DCOPs with dynamic objective functions would become easier to solve than DCOPs with fixed objective functions. In accordance with some recent studies, for example (Rand & Riolo 2005*b*) and (Kashtan *et al.* 2007), the experiment in this subsection provides another evidence of cases where the presence of dynamics might bring additional benefits to the evolutionary process. However, in our experiment the role and impact of dynamics in speeding up the search process are somewhat different from those in (Rand & Riolo 2005*b*) and (Kashtan *et al.* 2007). In (Rand & Riolo 2005*b*) and (Kashtan *et al.* 2007), the static problems are deceptive and due to that the algorithm might be trapped in a local optimum. When a change happens, the dynamic changes the problems, making them non-deceptive and hence indirectly helps the algorithms to escape from the current local optimum.

In our experiment, however, the static problem itself is not deceptive. The problem only becomes deceptive when some dynamics (moving constraints in this case) were introduced, changing the problem in a way that the changes go unnoticed by algorithms like HyperM. The interesting part lies in the fact that, when some more dynamic elements are introduced (dynamic objective function in this case) in addition to the existing dynamics, the problem becomes non-deceptive again! This is because the new dynamic triggers the diversity-introducing mechanism of HyperM, making the changes visible to HyperM to react. This experiment is an interesting

example where while the presence of one dynamic element might make the problem harder to solve, the occurrence of multiple dynamic elements might together make the problem easier to solve.

Table 5.9: This table shows the *triggered-time count* scores and the *detected-change count* scores of HyperM in a pair of problems with moving constraints exposing new optima after 11 changes (see Subsection 5.3.2 for descriptions).

Algorithms	G24_3 (NAO+fF)				G24_3b (NAO+dF)			
	Trigger Count		Detected Change Count		Trigger count		Detected Change Count	
	Value	stdDev	Value	stdDev	Value	stdDev	Value	stdDev
.HyperM-noElit	164.20	11.29	1.82	0.83	170.27	14.07	11.00	0.00
.HyperM_elit	0.00	0.00	0.00	0.00	30.00	0.00	11.00	0.00

NAO - Newly Appearing Optimum
fF / dF - fixed / dynamic objective Function

5.3.4 Possible suggestions to improve the drawbacks of current dynamic optimisation strategies in solving DCOPs

In the previous subsections, I have demonstrated that there are some difficulties when applying existing dynamic optimisation algorithms directly to solving DCOPs by combining the algorithms with the tested penalty methods. Some of the difficulties are caused by the use of the penalty methods, hence we can seek improvements by using different constraint handling techniques. However, others are caused by the nature of the dynamic optimisation strategies themselves and hence the strategies need to be modified.

Observations from the experimental results also suggest some suggestions on how to address the drawbacks listed in the previous subsections. First, based on our observation that elitism is useful for diversity-maintaining strategies in solving DCOPs, it might be useful to develop algorithms that support both elitism and diversity maintaining mechanism.

Second, given the fact that methods like HyperM are not able to detect changes because they mainly use change detectors (the best solution in case of HyperM) in the feasible regions, it might be useful to use change detectors in both feasible regions and infeasible regions.

Third, because experimental results show that tracking the existing optima might not be effective in certain cases of DCOPs, it might be useful to track the moving feasible regions instead. This is because after a change in DCOPs, the global optimum always either moves along with the feasible areas, or appears in a new feasible area. As a result, if we are able

to track feasible areas, we can increase the probability of tracking the actual global optimum. In the static case where the feasible regions do not move, tracking feasible regions still works because the algorithm can just focus on searching in the fixed feasible areas where the global optimum is located.

Finally, it might be useful to search in both feasible and infeasible regions. One common property of the penalty methods/values that I have tested in this section (see Subsection 5.3.2) is that feasible solutions always have higher fitness values than infeasible solutions and hence the algorithms might be biased more toward feasible solutions. It has been shown in the experimental results that such bias mechanism might make the algorithms less effective in solving DCOPs. Because of that, a good research direction might be to investigate the efficiency of other constraint handling techniques which are better in tolerating infeasible solutions. It would be interesting to see how other penalty functions with less bias feasibility/infeasibility would perform in solving DCOPs. Alternatively, it would also be interesting to study the performance of other non-penalty constraint handling methods, especially those that allow searching in the infeasible regions, in solving DCOPs.

One problem with choosing another existing constraint handling method for solving DCOPs is that, similar to the case of the tested penalty methods, other constraint handling techniques are also designed for stationary problems and have been tested in stationary problems only. Again we have the research question of whether the special characteristics of DCOPs would have any effect on these constraint-handling techniques and if there is, how can we improve that.

In the next sections, I will investigate the effect of the characteristics of DCOPs on some constraint handling techniques and then I will study how to improve any possible drawbacks.

5.4 Difficulties of some constraint handling strategies in solving DCOPs - an analysis

Because existing constraint handling (CH) strategies are designed for solving stationary problems and are tested in stationary problems only, there might be some difficulties in applying them to solving DCOPs, even if it is possible to combine them with existing dynamic optimisation (DO) strategies. The difficulties come from two main aspects: difficulties in handling environmental dynamics and difficulties in handling constraints.

In the following subsections, I will first review the two above types of difficulty in detail. After that, I will analyse some of these difficulties in experiments using one typical constraint handling technique - the repair method firstly proposed by Michalewicz and Nazhiyath (Michalewicz & Nazhiyath 1995) and implemented in (Michalewicz n.d.). It should be noted that in this section I do not attempt to provide a comprehensive review of existing constraint handling techniques (for this purpose readers are referred to recent survey papers, for example the studies and reports in (Michalewicz 1995, Back *et al.* 1997, Eiben 2001, Coello Coello 2002, Salcedo-Sanz 2009, Mezura-Montes 2009)). Instead, I will only study the possible difficulties of certain classes of constraint handling strategies, which are still widely used in recent applications, in solving continuous DCOPs.

5.4.1 Difficulties in handling dynamics

The most obvious reason for the difficulties in applying existing constraint handling (CH) strategies to solving DCOPs is the fact that these strategies are not designed to handle environmental dynamics. Consequently, they will not be able do the required tasks in dynamic optimisation such as detecting changes, tracking the moving optima, moving from one feasible region to another following the switch of the optima, and finding newly appearing optima. One might then raise the question of whether these difficulties can be overcome by combining existing CH strategies with existing DO strategies to gain the advantages from the two approaches and to alleviate the remaining disadvantages. Unfortunately, as will be shown below, not all difficulties can be resolved by combining existing CH strategies with existing DO strategies. In addition, that combination might also bring some new challenges due to the conflict of the optimisation goals of the two types of strategies.

Because existing CH and DO strategies are designed to satisfy two different goals, it is important to make sure that both goals are met when we combine a CH strategy with a DO strategy to solve DCOPs. In many cases, however, to meet both goals in one combination is not easy. This is because the goal of CH might conflict with the goal of DO and in some cases might even prevent the goal of DO - handling dynamics - from being accomplished. In this subsection I will discuss the possible difficulties that are caused by combining existing CH strategies with the following existing DO strategies: maintaining diversity, introducing diversity, and detecting changes based on performance drop.

Impacts on maintaining/introducing diversity

As already discussed, one of the important strategies in dynamic optimisation is to maintain or introduce diversity in the whole landscape to detect changes and to find newly-appearing optima or moving optima. However, when being combined with certain constraint handling techniques, the goal of diversity maintaining strategies may no longer be guaranteed. In other words, diversity might not be maintained in the whole landscape.

One of the reasons for this inefficiency is that in many constraint handling techniques, the original search space is specifically transformed so that the search algorithms only focus on certain areas instead of the whole original search space. In such cases, even if we use a diversity-introducing strategy such as HyperM to generate diversified individuals in the whole landscape, those diversified individuals that are generated in the unfocused areas might be neglected by the algorithms and hence do not contribute to the purpose of maintaining diversity. Typical examples of constraint handling strategies that adopt this landscape transformation approach are penalty methods where the constrained search space is transformed to an unconstrained landscape with penalised fitness values. Another example can be found in some approaches that use special representation/operators. In these approaches, the algorithms might be restricted to search only in the feasible regions, in a transformed feasible landscape, or in the boundaries of feasible regions. Detailed review and references for representative penalty approaches and special representations/operators approaches can be found in (Back *et al.* 1997, Coello Coello 2002).

In some other constraint handling techniques, individuals are selected not exclusively based on their actual fitness values but also on some special specifications. In such cases, the selection process is biased so that some types of individuals might have more probability of being selected than some others. For example, in Stochastic Ranking (Runarsson & Yao 2000) infeasible individuals might have more chances of being accepted depending on the given stochastic parameter. A counter example can be found in Simple Multimembered ES (Mezura-Montes & Coello 2005) where infeasible solutions are less likely to be accepted even if they have higher fitness values than the feasible ones. Another example is in a CH multi-objective approach (Venkatraman & Yen 2005) where individuals are ranked not entirely based on their original fitness values but also on the number of violated constraints. In constraint handling techniques like these, diversified individuals generated by dynamic optimisation strategies might not be selected in the same way

as they were originally designed for, i.e. the number of diversified individuals that are infeasible might become too large or too small. The way the diversity maintaining strategy work, as a result, might not be the same as it is in the unconstrained case.

Experimental evidence for the inefficiency mentioned in the cases above has already been shown in Subsection 5.3.3, where we can see that the diversity-maintaining/introducing strategies become less effective when combined with the tested penalty methods.

Impacts on change detection

Another possible difficulty of combining CH strategies with DO strategies is that the use of some existing constraint handling techniques might make change detection based on performance drop, a common DO technique, less effective. As already mentioned in the Subsection 5.3.2, algorithms like HyperM rely on the decrease in value of the fitness of the best solution over time to detect changes and to determine if the change is worth dealing with. The algorithm assumes that during the search process, if there is a degradation in the fitness values of the best solution found in each generation, there might be a change in the landscape and the previous found optimum might no longer the best optimum. However, when DO algorithms are combined with some CH techniques to solve DCOPs, such degradation in best fitness values might no longer be caused by an actual change in the landscape. Instead, the degradation might be caused either by an increase in penalty values (as in some dynamic/adaptive penalty methods) or by the elimination of the current good solutions from the population (as in some ranking-based methods). This problem of course might affect not only HyperM, but also any other change detection method that relies on monitoring the drop in fitness values of existing solutions.

One example can be found in some constraint handling techniques such as dynamic penalty or adaptive penalty e.g. (Joines & Houck 1994, Hadj-Alouane & Bean 1997, Hamida & Petrowski 2000), where the degradation of (modified) fitness values is not caused by environmental changes but by the increase over time of the penalty values. In these dynamic/adaptive penalty methods, initially the penalty values are low to tolerate more infeasible solutions. However, over time, based on the feedback of the algorithm or on the increased length of time, the penalty values are gradually increased to improve the convergence speed of the algorithm. The consequence of this dynamic/adaptive scheme is that if the detector solutions used by the change detection method is infeasible or becomes infeasible, over time their fitness value will decrease. Of course,

in penalty-based methods, if change detection is made on the original fitness values instead of on the penalised fitness values, the increase of penalty values will not have any impact on detecting changes. However, in this case, detecting changes based on the original fitness values might consequently suffer from another problem: changes in constraint functions will go undetected unless additional improvement is provided to make the method detect constraint changes explicitly.

In some other constraint handling techniques which use ranking-based methods (for example (Runarsson & Yao 2000, Mezura-Montes & Coello 2005, Venkatraman & Yen 2005)), the degradation of detectors' fitness values might be caused by the fact that during the selection process the current better solutions might be dropped in favour of other solutions. These solutions might have worse fitness value but are more useful for the constraint handling process. In these situations, there might also be a drop in the value of the best solutions at each generation.

The drop in fitness values of the detector solutions in both of the cases above might be wrongly considered by DO strategies like HyperM to be a change in the environment and this might consequently trigger the DO strategies to react inappropriately.

5.4.2 Difficulties in handling constraints

The difficulties of applying some existing CH strategies to solving DCOPs are caused not only by their possible weaknesses in handling dynamics (even when combined with existing DO strategies), but also by the fact that the ability of some CH strategies to handle constraints might also become less effective in DCOPs. This possible inefficiency is due to two reasons. First, the information that CH strategies have about the problem is not updated after each change. Second, the strategies themselves also do not update in accordance with changes in the environment. I will discuss these two reasons in detail below.

The issue of outdated information

One of the common difficulties for an algorithm to solve dynamic problems is that after a change all existing information that the algorithm has achieved or has been given about the problem might become outdated. If an algorithm continues to use its outdated knowledge about the previous problem to solve the newly changed problem, its performance might become less effective. For example, in algorithms using strictly feasible reference individuals like Genocop III (Michalewicz & Nazhiyath 1995, Michalewicz n.d.), after a change the population of feasible reference individuals found by the algorithm might no longer contain all feasible solutions

because the change has made some feasible solutions infeasible. Similarly, in some "decoder" methods the reference lists for ordinal representations (for example the ordered lists of cities (TSP (Grefenstette *et al.* 1985))/ ordered lists of knapsack items (KSP (Michalewicz 1997)) / order lists of tasks (scheduling (Syswerda 1991))) might no longer be in order after a change because the cities/items/tasks have changed their values. Another example can be found in dynamic/adaptive penalty methods e.g. (Hadj-Alouane & Bean 1997, Hamida & Petrowski 2000) where the penalty parameters learnt by the methods might no longer be suitable because the balance between feasible solutions and infeasible solutions has changed.

In order to resolve this difficulty, algorithms solving dynamic problems might need to be equipped with special mechanisms which allow them firstly to be able to detect the moment when a change happens and secondly to update their knowledge about the problem whenever a change happens. Because they are not specifically designed to handle dynamics, many of the existing CH strategies obviously do not have the necessary tools either to detect changes or to update their knowledge about the problem after changes. As already discussed in Subsection 5.4.1, one suggestion for improvement is to combine existing CH strategies with existing DO strategies. However, even if such combination is possible, there are still two remaining problems.

First, as discussed in subsection 5.4.1, even if we combine existing CH strategies with existing DO strategies, the task of detecting changes and updating problem information might still be difficult due to the special characteristics of some existing CH strategies. As already shown in Subsection 5.4.1, the implementation of some CH techniques might prevent existing DO strategies from handling the environmental dynamics effectively. The experimental results in Section 5.3.3 illustrate an example where the combination of existing DO strategies with penalty methods suffers from many difficulties in solving the tested DCOPs.

Second, there might be cases where the problem-specific information is given by users/designers and when a change happens, this information can only be updated by users/designers. The requirement of information being updated by users/designers, however, might not be applicable in many cases where the dynamic problems are solved online because users/designers might not know when a change happens and how a change happens. Examples of CH algorithms that require problem-specific information can be found in many decoder/repair/special-operator methods reviewed in (Coello Coello 2002) and (Salcedo-Sanz 2009).

All in all, the discussion above shows that their original designs might make it difficult for

some representative CH strategies to update their information/knowledge about the problem after a change happens. This shortcoming, in turn, makes these strategies work less effectively in handling the constraints in DCOPs. Later in subsection 5.4.4, I will analyse some experimental results showing how the issue of outdated information can affect the performance of one constraint handling technique - the repair method introduced in Genocop III (Michalewicz & Nazhiyath 1995, Michalewicz n.d.).

The issue of outdated strategy

Another difficulty of applying some of the existing CH strategies to solving DCOPs is that, even if they are able to update their knowledge/information about the problem after a change, they might still not be able to work most effectively because the strategies themselves are also outdated.

Strategy-being-outdated might occur when we use CH strategies that have problem-dependent parameters, whose values might be tailored to work best in only one (class of) stationary environment, to solve a DCOP. In such cases, if we fine-tune the parameter values for the problem before change, the algorithm might only work well until the moment when a change happens. If after a change the search landscape is represented by a problem of a different type, the strategy no longer works effectively. Typical examples of CH strategies that use problem-dependent parameters are penalty methods with pre-defined penalty factors and/or other pre-defined parameter that control how the penalty is defined (e.g. the target feasible ratio in the ASCHEA method (Hamida & Schoenauer 2002)). It has been reported (Smith & Coit 1997, Coello Coello 2002) that many penalty methods (static and dynamic) are sensitive to the values of penalty factors and/or other parameters, and that a parameter value that works well for one stationary problem might not work for another. Because of that, if different stages of a dynamic environment represent different problems, there might be no pre-defined penalty factor/parameter that can help the mentioned penalty methods to work well in that dynamic environment. Other examples of CH strategies that use problem-dependent parameters can be found in some combinatorial repair methods, methods with special operators, and decoder methods. A detailed review of these approaches can be found in (Coello Coello 2002) and (Salcedo-Sanz 2009)).

We believe that strategy-being-outdated might also occur with many adaptive CH strategies that are not problem-dependent in spite of the fact that they are considered as robust and/or

adaptive to solving different types of stationary constrained problems. As will be shown below, the reason for these classes of strategies becoming outdated in solving DCOPs is that they rely on some specific assumptions that are only true in stationary problems. In DCOPs these assumptions might become outdated once a change happens and consequently the corresponding strategies are also outdated.

Typical types of CH strategies in this class are self-adaptive fitness formulation (Farmani & Wright 2003) and the state-of-the-art method stochastic ranking (Runarsson & Yao 2000). The general approach of these strategies is to take feedbacks from the population during the search process and try to balance feasibility/infeasibility based on the performance of the current population, assuming that because the landscape is static, the feedback from the population always reflects a "memory" of information about the landscape and information about the convergence of the search process. In stationary constrained problems where initially the population covers the whole landscape and then gradually converges to specific areas, the above strategies can help to guide the population to converge to the correct global optimum in the feasible region. In dynamic environments, however, such strategies might become less effective because their assumption that the population always carries correct information about the landscape might no longer be true if a change happens. When a change happens, the search landscape might change its shape and consequently the "memory" of the population no longer reflects the property of the new landscape but only a small area where the population currently is. Even worse, the population might have already converged to a certain area and consequently the algorithm might not be able to explore other areas to find the new global optimum. Because strategies such as self-adaptive fitness formulation and stochastic ranking rely on the current population to handle constraints, if the current population cannot cover the search landscape properly these strategies will become less effective, or in other words become outdated with respect to the newly changed landscape.

Another type of CH strategies that rely on outdated assumptions are dynamic/adaptive CH strategies that rely on the running time value (e.g. the number of generations so far) to balance feasibility and infeasibility. CH strategies of this type e.g. (Joines & Houck 1994, Hadj-Alouane & Bean 1997, Hamida & Schoenauer 2002) also assume that the population can capture the information about the landscape. However, different from the strategies mentioned in the previous paragraph, runtime-relying strategies handle constraints by increasingly rejecting more

infeasible solutions as time goes by to increase the convergence speed to good regions. Again, because the assumption that the population can capture the property of the search space might no longer be true in dynamic environments, it follows that this approach might no longer be effective due to the occurrence of changes. For example, after a change in the landscape, the area to which the algorithm is converging might no longer contain the global optimum. In this case, if the CH strategy still imposes its previous balancing mechanism to increase convergence speed, the algorithm could end up converging to the wrong place and will not be able to track the moving optima.

Later in Subsection 5.4.4, I will analyse some experimental results showing how the issue of outdated strategy can affect the performance of a constraint handling technique, the repair method introduced in Genocop III (Michalewicz & Nazhiyath 1995, Michalewicz n.d.).

5.4.3 Possible suggestions to improve the drawbacks of current constraint handling strategies in solving DCOPs

The discussions in the two previous subsections show that, in order to handle constraints effectively in DCOPs, a constraint-handling strategy might also need to satisfy the requirements below:

1. It is necessary to make sure that the goal of handing constraints is not affected by the goal of handling dynamics. Particularly:

   (a) Diversified individuals might need to be distributed in all areas of the search space. In other words, CH strategies should not restrict those individuals generated for diversity purpose to only certain areas of the search space

   (b) Diversified solutions might need to be accepted at an acceptable rate or are introduced frequently to maintain diversity. In other words, CH strategies should not reject diversified individuals

   (c) Special attention might need to be made if change detection is undertaken by monitoring the fitness values of current individuals (when there is a drop of individual's performance, we need to check to see if the drop is really caused by an environmental change).

2. It is necessary to make sure that the algorithm is updated whenever a change happens. Particularly:

   (a) If an algorithm uses any knowledge about the problem to handle constraints, that knowledge needs to be updated whenever a change happens. This requires that in case the time of change is not known, an (implicit or explicit) change detection method needs to be implemented.

   (b) The constraint handling strategy might also need to be updated whenever a change happens.

   (c) The strategy should avoid using problem-dependent information because it might not be possible to update this type of information.

In order to work well in DCOPs, an algorithm needs to handle both environmental dynamics and constraints effectively. It means that a "good" algorithm for DCOPs needs to satisfy not only the requirements for handling constraints above but also the four requirements for handling dynamics identified in Subsection 5.3.4.

5.4.4 Experimental analyses

In this subsection I will carry out an experimental analysis to test the performance of the *repair method*, a representative CH strategy, in the G24 benchmark set. The purpose of the experimental study is to answer two questions. First, we would like to study the usefulness of the repair method in solving DCOPs. Second, we would like to verify if our hypotheses about the difficulties of DCOPs toward CH strategies, as mentioned in section 5.4.2, are true. In case the hypotheses are true I also would like to investigate how significant these difficulties would affect the performance of CH strategies (in particular the repair method in this case) in solving DCOPs. The result will help us to gain more understanding about how to design better algorithms to solve DCOPs.

Chosen constraint handling technique for the analysis

For this analysis I choose the *repair method* introduced by Michalewicz and Nazhiyath in (Michalewicz & Nazhiyath 1995), revised and implemented in (Michalewicz n.d.). There are four reasons to choose this CH technique. First, the method is representative. It represents a broad class of current CH techniques that do not use penalty functions but take feedbacks

from the search process to adaptively balance feasibility/infeasibility. The repair method also represents many current methods that use *repair operators* in solving constrained problems. I suspect that the repair method also has the major two possible drawbacks shared by existing CH strategies in handing constraints for DCOPs (as pointed out in section 5.3.2): outdated information and outdated strategy (as shown in Table 5.10). By testing this method, we will be able to verify whether these two drawbacks really bring any negative effect to existing CH strategies, and how significant is the effect.

Second, we believe that the repair method has some traits that make it more robust than some other CH techniques in solving DCOPs. Table 5.10 shows that the *repair method* satisfies many of the requirements suggested for solving DCOPs, and especially it is possible to modify the method to satisfy all requirements. I am interested in how the special characteristics of DCOPs would affect such a robust CH technique. I am also interested in modifying robust techniques like the repair method to work better in DCOPs. By choosing the repair method for this experiment, we will have more knowledge to answer these two questions.

Third, the repair method is simple, easy to implement and is integrated with a GA-based algorithm (Genocop III). Using the method makes it easier to compare its performance with the existing DO algorithms: GA / RIGA / HyperM that I have tested in Subsection 5.3.3. Using the method also makes it easier to integrate the method with other DO and CH strategies to develop new algorithms for solving DCOPs.

Finally, different from most other repair mechanisms, the repair method proposed in (Michalewicz n.d.) is problem-independent and is designed specifically for the continuous domain. This characteristic makes the method perfectly suitable for our purpose of testing CH strategies in our continuous benchmark set G24. The method also facilitates us in developing new algorithms to solve continuous DCOPs in future research.

Table 5.10: In order to solve DCOPs effectively, a constraing handling method might need to satisfy a number of requirements as suggested in subsections 5.3.4 and 5.4.3. This table shows us how many requirements have been satisfied by the *repair method* implemented in (Michalewicz n.d.), and how many requirements can be satisfied if we modify the method.

Requirements	Satisfy?	Modifiable?	How does it satisfy? / Why is it modifiable to satisfy?
Suggested requirements to handle dynamics in DCOPs (Subsection 5.3.4)			
Elitism	Yes	Yes	The reference population contains elitist members
Diversity maintaining	Partly	Yes	The method itself has good diversity. In addition, its mutation can be modified to increase diversity
Search in feasible and infeasible region	Yes	Yes	The search population accepts both feasible and infeasible individuals provided that they can offer good repaired solutions
Track the moving feasible regions	Partly	Yes	The repairing process is able to produce feasible individuals for the tracking process
Change detection in feasible and infeasible regions	No	Yes	The method can be integrated with a change detection mechanism
Suggested requirements to handle constraints in DCOPs (Subsection 5.4.3)			
Diversified individuals cover the whole search space	Partly	Yes	The method does not restrict its search operation to any specific area nor transform the original landscape to a limited search space
Diversified individuals are retained/ re-introduced frequently	Partly	Yes	Diversified individuals are accepted regardless of their feasibility provided that they can offer good repaired solutions
Drop in best fitness values are caused by environmental changes only?	Yes	Yes	In the latest version (Michalewicz n.d.), best fitness values are also the best feasible objective values. Because of that, if there is any drop in the best fitness values, it means that the objective function changes.
Update problem information after each change?	No	Yes	The algorithm takes feedback from the population to update its information. By appropriately updating the population we will be able to update the algorithm
Update CH strategy after each change?	No	Yes	The method takes feedback from the population to update its information. By updating the population we will be able to update the algorithm
Use problem-independent information only?	Yes	Yes	The method only use problem-independent information

Repair algorithms & the repair method in Genocop III

General ideas Repairing infeasible solutions is a common approach widely used in many different EAs to solve combinatorial and continuous constrained problems. The idea is that, if it is possible to map (repair) an infeasible solution to a feasible solution, then instead of searching the best feasible solution directly, it might be possible to look for an individual that can potentially produce the best repaired solution. To implement such a type of search, it is necessary to change the way the fitness value of an individual is calculated. Instead of being based on its objective value, now the fitness value of an individual is calculated based on the quality of the corresponding repaired solution. The better the repaired solution, the higher the fitness value of an individual. In certain cases, the feasible solution created by the repair process can also be used to replace some of the search individuals.

In general, a repair approach can be represented in three steps as follows:

1. If a newly created individual $\mathbf{s}$ (can be feasible or infeasible) needs repair, use a heuristic *repair* () to repair $\mathbf{s}$, mapping $\mathbf{s}$ to a new, feasible individual $\mathbf{z}$.

2. The objective value $f(\mathbf{z})$ of $\mathbf{z}$ then is used as input to calculate the fitness value of $\mathbf{s}$, i.e. $\text{eval}(\mathbf{s}) = h(f(\mathbf{z}))$ where h is the mapping from objective values to fitness values.

3. If the repair approach is Lamarckian, replace one or some search individuals by $\mathbf{z}$

The *repair method* used in this experiment was firstly proposed in (Michalewicz & Nazhiyath 1995) and was integrated as a part of the Genocop III algorithm (Michalewicz n.d.), which is designed for continuous domain. This method also follows the general three steps given above, in which the *repair* () heuristic can be described as follows:

1. The population is divided into two sub-populations: a search population S containing normally-evolving individuals, which can be fully feasible or only linearly feasible, and a reference population R containing only fully feasible individuals.

2. During the search process, while each individual $\mathbf{r}$ in the reference population R is evaluated using their objective function as usual, each individual $\mathbf{s}$ in the search population S is considered to be repaired based on an individual from R. Detail of the repair routine can be found in the prosecode of the routine *Repair* in Algorithm 9 (page 151).

It is important to note that there are two possible variants of deciding whether a search individual $\mathbf{s}$ needs to be repaired in Genocop III (step 2 above). In the first variant described in (Michalewicz & Nazhiyath 1995), it is stated that a search individual $\mathbf{s}$ should be repaired *only* if $\mathbf{s}$ is infeasible. In such case, after the repair process the fitness value of $\mathbf{s}$ will be equal to the fitness value of the mapped solution $\mathbf{z}$ (see the three-step procedure above), i.e. $\text{eval}(\mathbf{s}) = \text{eval}(\mathbf{z}) = h(f(\mathbf{z}))$. If $\mathbf{s}$ is feasible, it will still have its original fitness value, i.e. $\text{eval}(\mathbf{s}) = h(f(\mathbf{s}))$. However, in the latest version of Genocop III provided by the authors (Michalewicz n.d.), the implementation of the algorithm shows that search individuals are repaired in any case regardless of their feasibility. This difference in implementations leads to different ways of selecting individuals in the two variants. In the first variant, search individuals have more chance to be selected for the next generation if they either found a good feasible solution or produced a good repaired solution. In the second variant, the way individuals are selected is only based on their performance in producing good repaired solution.

In all experiments in this chapter, I choose to implement the second variant of the repair method, i.e. to repair search individuals regardless of their feasibility. I choose this variant because it is implemented in the official source code provided by the authors. In addition, this variant is the latest version and hence should supposedly be better than the earlier version. Experiments on the first variant of the repair method will be carried out in our future investigations.

From now on, in this chapter unless stated otherwise we will use the term *repair method* to refer to the continuous-based repair approach proposed in (Michalewicz n.d.).

Feasibility/infeasibility balancing strategy and problem knowledge in the repair method Before taking analysis on the repair method, we need to understand the strategy that the method uses to balance feasibility/infeasibility, and the type of problem information that the method uses to guide the strategy in solving constrained problems.

Repair method and other repair approaches have the ability to adaptively balance feasibility and infeasibility. This balance is achieved by two procedures. First, the method accepts both infeasible individuals and feasible individuals, provided that they can produce good repaired solutions. This is because individuals are evaluated not based on their actual objective values or feasibility, but on how good the feasible, repaired solutions that they produce are. This is

accomplished by updating the fitness value eval (**s**) of each search individual **s** with the objective value $f(\mathbf{z})$ of the repaired solution **z** i.e. $\text{eval}(\mathbf{s}) = h(f(\mathbf{z}))$. Second, by updating the fitness values of search individuals like that, the repair method ensures that while infeasible solutions are accepted, they cannot have better fitness value than the best feasible solution available.

In order to work effectively, the strategy above needs certain type of problem information, which is provided by the reference population and the search population. The reference population is an essential source of information for the balancing strategy to direct the algorithm toward promising feasible regions. This is because, during the repair process (see the *Repair* routine in Algorithm 9, page 151), newly repaired solutions are always generated in the lines toward reference individuals (for each search individual, repaired solutions are generated in the segment between that search individual and a reference individual). The reference individuals also provide the balancing strategy with information about the best feasible solution available (via their fitness values) so that the strategy can make sure that no infeasible individual can have better fitness values than that.

The search population is also an essential source of problem information for the balancing strategy. This is because the fitness values of search individuals help to indicate which point in the landscape would lead to the potentially promising feasible region (via the repair process). In the selection phase the balancing strategy then uses that information to select those individuals that would potentially lead to the most promising regions.

How can the characteristics of DCOPs affect repair method? As mentioned earlier, although the repair method has some advantages in solving DCOPs as shown in Table 5.10, I suspect that the method still suffers from the problem of outdated information, which in turn makes the feasibility/infeasibility balancing strategy become outdated.

The first type of information that might become outdated when a change happens is the fitness values of search individuals. Because the fitness value of a search individual is always based on the objective value of the feasible solution repaired by that individual, it is assumed that the search population always offers a "memory" of good areas in the landscape and directions toward these good areas. The direction can be interpreted as: the higher the fitness value of an individual, the better the feasible region we can get by repairing that individual.

This assumption is true in stationary environment because the "memory" is never outdated.

However, in dynamic environment, the memory, or fitness values of search individuals, can become outdated right after a change if the objective values of the corresponding repaired solutions change. Particularly, the high fitness values of existing individuals might no longer lead to good repaired solutions and vice versa. Even worse, search individuals with high-but-outdated fitness values might even wrongly bias the selection process and hence make the search process less effective. Search individuals can only become updated if we keep repairing them at every generation. However, in the repair method this is not always the case because not all individuals are selected for the repairing process at each generation.

The second type of information that might become outdated when a change happens is the reference individuals, which are used to repair all other search individuals. The key assumption in the repair method is that all reference individuals are feasible and are the best in the population. This assumption is only true in stationary environments. In dynamic environments, after a change happens, some existing reference individuals might no longer remain the best in the population or might even become infeasible. These outdated reference individuals not only violate the assumption named above but might also wrongly bias the search and drive more individuals *away* from the good regions, making the search process less effective. Because the reference individual only evolves after a certain period (100 function evaluations as in (Michalewicz n.d.)), if there is any change occurs during that period, reference individuals are likely to become outdated.

In the following experiments I will analyse if our hypotheses about the effects of DCOPs' characteristics on repair methods are correct and how significant the effects are. The experiments will also help us to verify if our hypothesis about the usefulness of the repair method in solving DCOPs is true.

Test settings

Tested algorithms Although in (Michalewicz & Nazhiyath 1995) and (Michalewicz n.d.) the repair method is integrated with Genocop III, this method is algorithm-independent and can be integrated with many different continuous evolutionary optimisation algorithm. In the experiment in this section, I choose to integrate the repair method with basic GA. The integrated version is called GA+Repair and is described in detail in Algorithm 8 (page 150). There are two reasons to choose GA+Repair as the tested algorithm in this experiment. First, GA+Repair

makes it possible to analyse the strengths and weaknesses of the repair strategy. In the original version (Genocop III), it is difficult to analyse if the reason for any increase/decrease in performance is due to the repair method because Genocop III implements multiple CH strategies (beside the repair operator, there are ten other specialised operators to handle linear constraints). By integrating just only the repair operator with basic GA in GA+Repair, it would be easier to analyse the effect of the repair method: any difference in performance between GA+Repair and basic GA would be caused by the repair operation.

Second, GA+Repair makes it possible to compare the usefulness of the repair method in solving DCOPs with other DO strategies previously tested in this chapter. Because all other strategies are tested when they are integrated with basic GA, it is natural that in order to compare the repair method with these strategies we should also integrate the method with GA.

Because the purpose of GA+Repair is to evaluate the repair operation only, the algorithm is significantly simpler than Genocop III although both algorithms use exactly the same repair operator. The differences between GA+Repair and Genocop III are: (1) GA+Repair does not have any specialised method to handle linear constraints like Genocop III. Because of that, individuals in the search population are not required to satisfy linear constraints as in Genocop III. GA+Repair also does not require that nonlinear and linear constraints have to be treated differently as in Genocop III; (2) GA+Repair only has two normal GA operators: crossover and mutation compared to ten specialised operators in Genocop III; and (3) while Genocop III allows about 25% of the repaired individuals to replace individuals in the population (Lamarckian evolution), in GA+Repair none of the repaired individual is used to replace the original individuals (Baldwinian evolution). The reason is that according to the study in Subsection 6.4.5 and in (Nguyen & Yao 2010*b*), we found that the use of Lamarckian evolution does not significantly increase or decrease the performance of Genocop III in solving DCOPs.

Despite the difference above between GA+Repair and Genocop III, I observe that both algorithms have very similar behaviours when solving different groups of DCOPs in the G24 benchmark set. It means that Genocop III also shows the same advantages and disadvantages as GA+Repair when solving DCOPs, except that Genocop III has an overall better performance when handling constraints thanks to the additional CH methods. The similarity in behaviours of GA+Repair and Genocop III suggests that the result tested with GA+Repair can be generalised to other approaches that use the repair method. For a detailed results of Genocop III's

performance in the G24 benchmark set and a comparison of its performance with other existing and new algorithms, readers are referred to our other study in Chapter 6 and in (Nguyen & Yao 2010*b*).

Parameter settings The tested algorithms use the same parameter settings as the previously tested GA, RIGA, and HyperM except that the population now is divided into a search population and a reference population (see Table 5.6, page 113), as implemented in the original Genocop III algorithm (Michalewicz n.d.).

Performance measures To carry out the analysis, I use three different types of measures. The first measure, which is our modified version of the *off-line error* measure (see Subsection 5.3.2), is used to evaluate/compare the general performance of the GA+Repair. Similar to the previous experiment, using this measure I will also firstly summarise the average performance of GA+Repair in each major group of problems (see test results in Figure 5.7, page 151) and secondly investigate the effect of each problem characteristic on GA+Repair by analysing their performance in 21 test cases shown in Table 5.5 of Section 5.2 (see test results in Figure 5.4, page 122 and Figure 5.5, page 123).

The second and third measures, which are our newly proposed measures, are both used to analyse the behaviour of the repair method in DCOPs. The second measure, named *plot of number of reference individuals that are feasible*, is used to analyse the behaviour of the *repair method* when some reference individuals become outdated due to environmental changes - see Figure 5.8 (page 155). The third measure, named *plot of number of feasible individuals in each disconnected feasible region,* is used to analyse the ability of repair methods to balance feasibility and infeasibility in problems with optima switching between disconnected feasible regions - see Figure 5.9 (page 156). Details of these two measures will be described in the following subsections.

The impact of outdated information/strategy on the performance of the repair method

Overall observation of performance in groups of problems To analyse the overall performance of the repair method, I firstly study the average performance of GA+Repair in each major group of problems (Figure 5.7) and then study the performance of the algorithm in pair of problems with different characteristics (Figure 5.4, page 122 and Figure 5.5, page 123). The

Algorithm 8 GA+Repair

Note: It is assumed that the problem is maximisation

1. *Initialise*:
   (a) Randomly initialise m individuals in search pop S
   (b) Initialise n individuals in the reference population R
      i. Randomly generate points until a feasible $\mathbf{r}$ is found
      ii. Update the fitness value of r: eval $(\mathbf{r}) = f(\mathbf{r})$ and add r to R
      iii. Repeat step 1(b)i until n individuals are found
2. *Search*: For $i = 1 : m$
   (a) $p_1 = U(0,1)\,; p_2 = U(0,1)$
   (b) *Crossover*: **If** $(p_1 < P_{Xover})$
      i. Use nonlinear ranking selection to choose a pair of parents from S
      ii. Crossover an offspring $\mathbf{s}$ from the chosen parents
      iii. Evaluate $\mathbf{s}$ and repair $\mathbf{s}$ using the routine $Repair(\mathbf{s})$
      iv. Use nonlinear ranking selection to replace one of the worst individuals in S by $\mathbf{s}$
   (c) *Mutation*: **If** $(p_2 < P_{Mutate})$
      i. Use nonlinear ranking selection to choose a parent from S
      ii. Mutate an offspring $\mathbf{s}$ from the chosen parent
      iii. Evaluate $\mathbf{s}$ and repair $\mathbf{s}$ using the routine $Repair(\mathbf{s})$
      iv. Use nonlinear ranking selection to replace one of the worst individuals in S by $\mathbf{s}$
   (d) *Otherwise*: **If** $(p_1 \geqslant P_{Xover})$ **and** $(p_2 \geqslant P_{Mutate})$
      i. Use nonlinear ranking selection to choose an individual $\mathbf{s}$ from S
      ii. If $\mathbf{s}$ has not been evaluated since the last generation, evaluate $\mathbf{s}$
      iii. Repair $\mathbf{s}$ using the routine $Repair(\mathbf{s})$
      iv. Using nonlinear ranking selection to replace one of the worst individuals in S by $\mathbf{s}$
3. Evolve the reference population after each 100 evaluations: For $i = 1 : n$
   (a) *Crossover*: **If** $(U(0,1) < P_{Xover})$
      i. Use nonlinear ranking selection to choose a pair of parents from R
      ii. Crossover an offspring $\mathbf{r}$ from the parents
      iii. **If** $\mathbf{r}$ is feasible
         A. Evaluate $\mathbf{r}$ and $\mathbf{x}$,the better of the two parents
         B. If $f(\mathbf{r})$ better than $f(\mathbf{x})$ then $\mathbf{x} = \mathbf{r}$ and fitness value eval $(\mathbf{x}) = f(\mathbf{r})$
   (b) *Mutation*: **If** $(U(0,1) < P_{Mutation})$
      i. Use nonlinear ranking selection to choose a parent $\mathbf{x}$ from R
      ii. Mutate an offspring $\mathbf{r}$ from $\mathbf{x}$
      iii. **If** $\mathbf{r}$ is feasible
         A. Evaluate $\mathbf{r}$ and $\mathbf{x}$
         B. If $f(\mathbf{r})$ better than $f(\mathbf{x})$ then $\mathbf{x} = \mathbf{r}$ and fitness value eval $(\mathbf{x}) = f(\mathbf{r})$
4. Return to step 2

Algorithm 9 *routine* Repair(Indiv **s**)

1. Randomly pick an individual $\mathbf{r} \in R$
2. Generate individual $\mathbf{z}$ in the segment between $\mathbf{s}$ and $\mathbf{r}$
   (a) $a = U(0, 1)$
   (b) $\mathbf{z} = a.\mathbf{s} + (1 - a).\mathbf{r}$
   (c) While $\mathbf{z}$ is infeasible, back to step 2a
   (d) If a feasible z is not found after 100 trials, $z = r$ and $\text{eval}(\mathbf{z}) = \text{eval}(\mathbf{r})$
3. (a) Evaluate $\mathbf{z}$
   (b) If ($f(\mathbf{z})$ better than $f(\mathbf{r})$) then $\mathbf{r} = \mathbf{z}$; $\text{eval}(\mathbf{r}) = f(\mathbf{z})$
   (c) Update the fitness value of $\mathbf{s}$: $\text{eval}(\mathbf{s}) = f(\mathbf{z})$
4. Return the individual $\mathbf{s}$

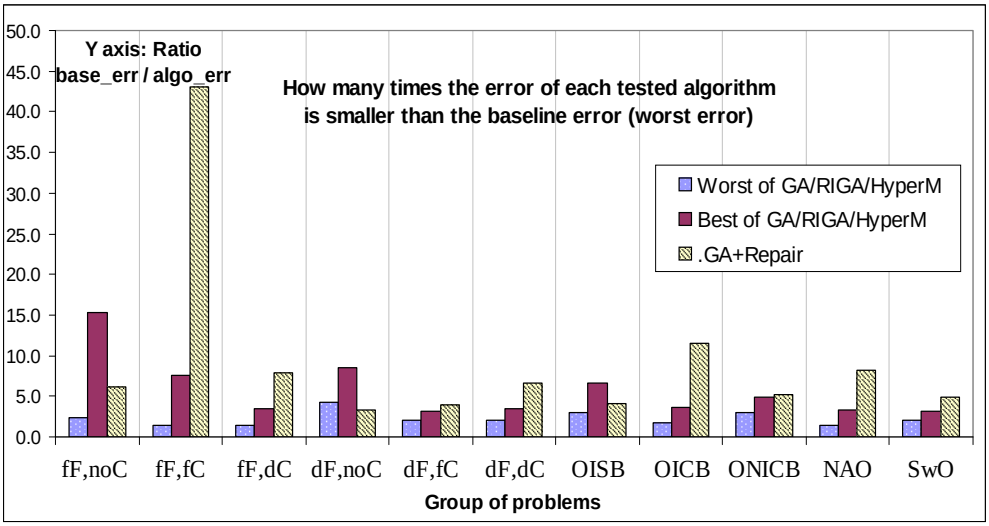

Figure 5.7: This figure shows the performance of GA+Repair compared with the worst and best performance of existing dynamic optimisation algorithms (GA, RIGA and hyperM) in different groups of problems. As in Figure 5.3, algorithms' performance is evaluated based on how many times they are better than the base-line error, which is the worst (largest) error among all algorithms. This score is represented in the vertical axis. The horizontal axis shows different group of problems. Explanations for the abbreviations of problem groups can be found in the caption of Figure 5.3.

performance in groups generally confirm our hypotheses about the advantages and disadvantages of the repair method in solving DCOPs.

In the group of stationary constrained problems (fF, fC), the results in Figure 5.7 show that as expected, a specialised CH technique as the repair method in GA+Repair is much more useful

than methods not designed for handling constraints as existing DO algorithms. GA+Repair performs significantly better than the existing DO algorithms by factors of 6.4 to 45.44. In stationary unconstrained group (fF, noC), also as expected the repair method in GA+Repair is no longer particularly useful. Figure 5.7 shows that GA+Repair also performs worse than all other methods in dynamic, unconstrained problems (dF, noC).

In the groups of DCOPs (fF+dC, dF+fC, dF+dC), things are different. Although GA+Repair works very well in stationary constrained cases as mentioned above, the algorithm becomes less successful in the dynamic cases. As can be seen in Figure 5.7, in DCOPs the difference between GA+Repair and GA is no longer as significant as it is in the stationary constrained case, meaning that the performance of GA+Repair significantly decreases (by factors of 5.2 to 41.3). This happens in all three cases of DCOPs: where only the constraints are dynamic (fF, dC); where only the objective functions are dynamic (dF, fC) and where both constraints and objective functions are dynamic (dF, dC).

Details of the impact of dynamic objective function on the repair method can be seen in the pair-wise comparisons in plot c and plot d of Figure 5.4 where GA+Repair is tested in pair of almost identical constrained problems except for that one has a fixed objective function and the other has a dynamic objective function. As can be seen in these plots, the performance of GA+Repair is significantly decreased in case the objective function is dynamic. The difference in performance of GA+Repair between the two problems of each pair is significantly larger than that of GA and existing DO algorithms, meaning that the presence of dynamic objective function has a much greater impact on repair method than on GA and existing DO methods.

Details of the impact of dynamic constraints on the repair method can be seen in the pair-wise comparisons in plot i of Figure 5.4 and plot a of Figure 5.5 where GA+Repair is tested in pair of almost identical constrained problems except for that one has fixed constraints and the other has dynamic constraints. Similar to the previous case, the results also show that the performance of GA+Repair is significantly decreased in case the constraints are dynamic and that the presence of dynamic constraints has a much greater impact on repair method than on GA and existing DO methods. This significant impact of DCOP's environmental dynamics on GA+Repair's performance proves our hypothesis that repair method suffers from difficulties in solving DCOPs.

However, although the presence of environmental dynamics does significantly degrade the

performance of GA+Repair, in Figure 5.7 it is interesting to see that the algorithm still has better performance than existing DO algorithms (only that the difference become significantly smaller in the dynamic cases). This observation proves our hypothesis that the repair method has some characteristics that make it very promising for solving DCOPs.

Another interesting, and somewhat counter-intuitive observation in our experiment is that the presence of constraints do not make the problems more difficult to solve by GA+Repair. Instead, I found that the presence of constraints always help GA+Repair to work better. Evidence can be found in the pair-wise comparison in plots a, e, f, j, k, l of Figure 5.4 and in plot h of Figure 5.5 where GA+Repair always has better performance in the problem with constraints than in the problem without constraints. The experiment also shows that GA+Repair has better performance in case there is an infeasible barrier separating two feasible regions, and that the larger the barrier, the better the performance of GA+Repair (see plot d, e in Figure 5.5). I found that these two behaviours are due to the nature of the repair method in handling constraints. This is also one of the reasons why we believe that repair method has some advantages in solving DCOPs. A detailed analysis of this behaviour will be provided in Chapter 6.

The experimental results above confirms that the presence of dynamic does have a significant effect on the performance of the repair method. Now it would be interesting to see if that effect is indeed caused by the outdated problem information (reference individuals and search individuals) and by the outdated balancing strategy as suspected in our hypothesis. In order to answer this question, I will undertake a further analysis as can be seen below.

Analyse the behaviours of outdated reference individuals As recalled in subsection 5.4.4, I suspected that the reason for GA+Repair to work less effective in dynamic constrained problems is that the algorithm might have outdated information and its strategy might also become outdated. One type of outdated information is outdated reference individuals, in which some members of the reference population might have their objective values changed or even become infeasible after a change. Because the core idea of the repair method is based on the reference individuals, if these individuals are not updated, the algorithm would not be able to run correctly.

To test if the algorithm is able to update the reference individuals properly, I use our proposed measure: *plot of number of reference individuals that are feasible* (mentioned in subsection 5.4.4).

If the algorithm is able to update the reference individuals properly, it should be able to maintain a reference population of all feasible individuals all the time during the search process.

The most suitable environments to test this behaviour of repair method are DCOPs with dynamic constraints where after each change the previous best feasible solutions are hidden by the moving infeasible region. In the G24 benchmark set, the problems that have this property are the G24_4, G24_5 and G24_7. G24_4 and G24_5 belong to the problem group *dF,dC* while G24_7 belongs to the problem group *fF,dC* (see Figure 5.7). As discussed earlier, in both groups the performance of GA+Repair decreases significantly compared to the case where the constraints are fixed (fF,fC).

In this analysis we will see if the moving infeasible region makes any of the reference individuals to become infeasible. If no reference individual becomes infeasible, after each change the total number of feasible reference individuals should remain to be five. If one or more individuals do become infeasible, there should be a drop in the total number of feasible reference individuals. In that case, it is likely that reference individuals being outdated is one of the reason that make the repair method works less effectively in G24_4, G24_5 and G24_7.

The *plot of number of reference individuals that are feasible* of GA+Repair is given in Figure 5.8. The figure shows that, in all cases the original repair method is not able to keep all reference individuals feasible during the search. When a change happens, the number of feasible reference individuals drops to a very low level. Although the algorithm is able to slowly recover from the drop (i.e. the number of individuals that are feasible increases over time), in most of the time the number of feasible reference individuals is much lower than five. This violates the requirement of the original repair method that the reference population needs to contain only feasible solutions.

The results confirm our hypothesis that after a change, the algorithm's problem information, which in this case is the population of reference individuals, has become outdated and consequently might wrongly bias the algorithm to wrong directions. The reason for this behaviour is that after each change, the infeasible regions moves and hide the currently best region, which contains most of the reference individuals. This makes these individuals become infeasible.

Analyse the behaviours of the outdated balancing strategy In Subsection 5.4.4, I also suspected that individuals-being-outdated can also have a negative impact on the balancing

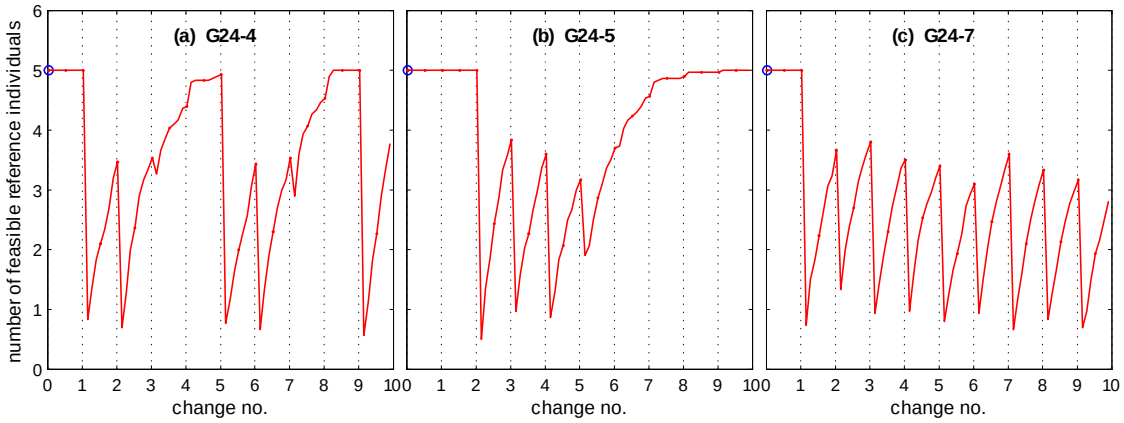

Figure 5.8: This figure shows how GA+Repair maintains feasible reference individuals in problems with moving infeasible regions. The total number of reference individuals is five. The plot in the figures shows, among these five reference individuals, how many are actually feasible during the search process. As can be seen, GA+Repair is not able to keep all reference individuals feasible during the search. Instead, after each change the number of feasible individuals drops to a very low level.

strategy, which balances feasibility and infeasibility, of the repair method. To test if the algorithm is still able to balance feasibility/infeasibility properly in dynamic environments, I use our proposed measure: *plot of number of feasible individuals in each disconnected feasible region* (mentioned in subsection 5.4.4) to monitor the number of feasible individuals in each disconnected feasible region and the ratio of feasibility/infeasibility. If the balancing mechanism works well in the DCOP case, it should be able to manage a good distribution of individuals so that the better feasible regions should have more feasible individuals.

The most suitable environments to test this behaviour of existing repair method are DCOPs with two disconnected feasible regions where the global optimum keeps switching from one region to another after each change or after some consecutive changes. In the G24 benchmark set, the problems that have this property are the G24_1, G24_2, G24_3b, G24_4, G24_5, G24_6a, G24_6c, G24_6d, and G24_8b where the global optimum switches from one region to another after each period of one or two changes. All these problems belong to the group *SwO* in Figure 5.7 (page 151), where we can see that the performance of GA+Repair significantly decreases compared to the stationary constrained case (fF, fC). In such SwO problems like these, if the balancing mechanism of GA+Repair works well, at each period between changes the algorithm should be able to focus most feasible individuals on the region where the global optimum currently is while still maintain the same ratio of feasibility/infeasibility for diversity

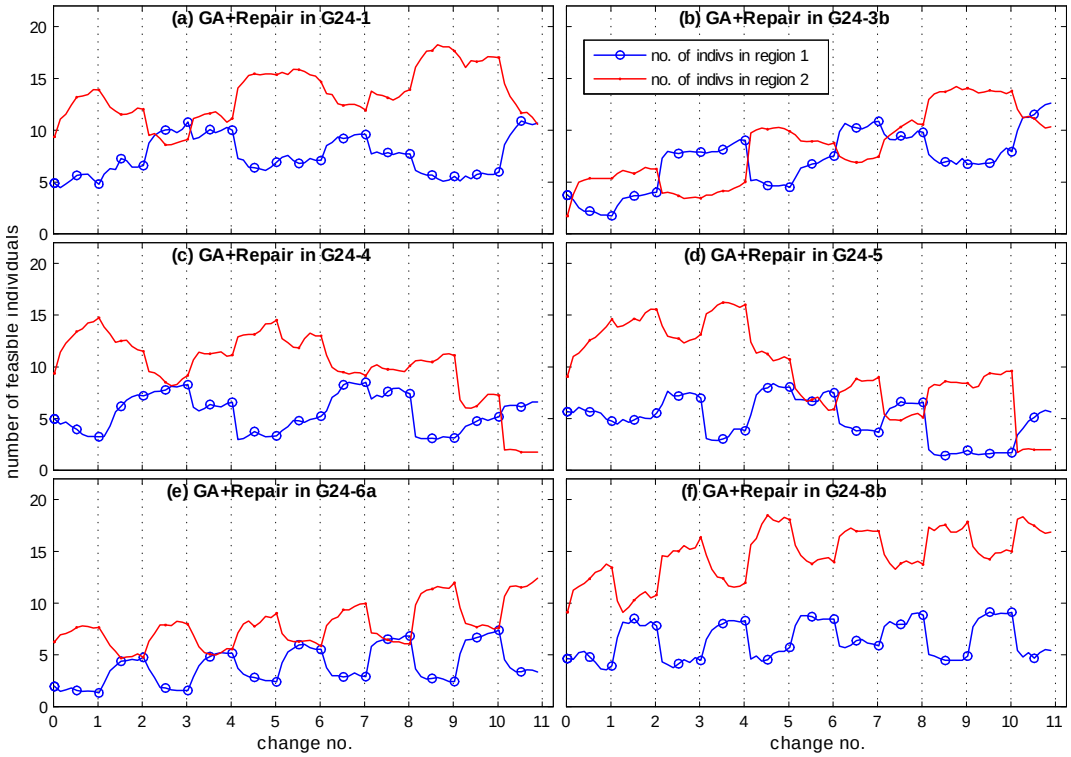

Figure 5.9: This figure shows how the balance strategy of GA+Repair distributes its feasible individuals in disconnected feasible regions. The problems tested in this figure are those with global optima switching between two disconnected feasible regions. The plot lines with circles show the number of feasible individuals in region 1, and the plain plot lines show the number of feasible individuals in region 2. If the balance strategy works well, most individuals should be focused on the region where the global optimum is currently in. It means that when the optimum switches to region 2, the number of individuals in region 2 should be high and the number of individuals in region 1 should be low. When the optimum switches back to region 1, the reverse thing should happen, i.e. number of individuals in region 1 should be high and that number in region 2 should be low.
The result shows that GA+Repair is not able to focus most of its individuals to the appropriate region. Instead, the majority of individuals still remained in one single region (region 2), which is where the optimum firstly was.

purpose. Otherwise, the outdated balancing-strategy might be one of the reasons that make the repair method become less effective in SwO problems.

The *plot of number of feasible individuals in each disconnected feasible region* of GA+Repair in these functions is given in Figure 5.9. It should be noted that the measure scores of GA+Repair in three problems: G24_2, and G24_6c/d are not shown in Figure 5.9 because they are the same as that of G24_5, and G24_6a, respectively.

Figure 5.9 shows that in all cases except G24_3b, the repair method is not able to focus most feasible individuals on the region where the global optimum is currently in. Instead, the

majority of feasible individuals still remained in one single region (region 2), which is where the global optimum firstly was before the changes happen. The number of individuals in the other region (region 1) remains low regardless of whether the global optimum has switched into the region or not. These results show that, due to its outdated information and strategy, the algorithm is not able to follow the switching optimum well.

The reason for this behaviour is that, initially most of the individuals were in region 2 because it is where the global optimum was firstly in. After the first change, although the global optimum has switched to region 1, many individuals in region 2 were not updated, and hence still had their old and outdated fitness values, which might be even higher than the new, after-change global optimum fitness value. These incorrect but high fitness values cause the outdated individuals to continue dominating the population and attract a large number of individuals to the old feasible region 2 despite the lack of the actual global optimum there.

It should also be noted that in solving problems in the G24 benchmark set, individuals being outdated might not always be totally harmful because the changes in many problems are cyclic. It means that although many infeasible or poor reference individuals are retained due to their outdated high fitness values, to some extent these outdated individuals might become useful in future changes because the global optimum might re-appear in previous places. In such cases, the outdated individuals might actually play the role as memory elements and hence might help the algorithm to recall the previous good solutions in cyclic problems. However, it is not clear of how much benefit such memory elements could bring, because our experiments show that GA+Repair still becomes less effective in the presence of environmental dynamics. In addition, it is obvious that such type of memory elements would not be useful in problems with no cyclic dynamics.

Summary

In summary, the experiments in this section generally confirm our hypotheses about the advantages and disadvantages of the repair method in solving DCOPs. First, the experimental results show evidence that the repair method might be useful for solving dynamic constrained problems. Second, the results also show evidence that outdated information and outdated balancing strategy can make such constraint handling strategies as the repair method less effective.

5.5 Summary

5.5.1 Summary of contributions

The contributions of this chapter can be summarised as follow:

1. *New investigations on the unknown characteristics of DCOPs*: Some special, not-well-studied characteristics of DCOPs that might cause significant difficulties to existing DO and CH strategies were identified for the first time.

2. *New developments of new benchmark problems and performance measures*: 18 new benchmark problems (22 pairs) and seven new performance measures were developed. One existing measure was also modified to be usable in DCOPs.

3. *New investigations on the strengths and weaknesses of existing DO strategies (GA/RIGA/HyperM) and CH strategies (repair methods) in solving DCOPs.* The experimental analyses reveal some interesting findings, which can be categorised in three groups as follows:

   (a) *The performance of existing DO strategies in DCOPs*:

      i. The use of elitism might have a positive impact on the performance of existing diversity-maintaining strategies. Elitism however might also have a negative impact on the performance of diversity-introducing strategies if they are not used in combination with diversity-maintaining strategies.

      ii. The presence of infeasible areas has a negative impact on the performance of diversity-introducing/diversity-maintaining strategies.

      iii. The presence of switching optima (between disconnected regions) has a negative impact on the performance of DO strategies if they are combined with penalty functions.

      iv. The presence of moving infeasible areas has a negative impact on the performance of tracking-previous-optima strategies.

   (b) *The performance of some existing CH strategies in DCOPs*: Even if we can combine CH strategies with DO strategies, there might be two types of difficulties:

      i. Difficulties in handling dynamics, in particularly maintaining diversity and detecting changes.

ii. Difficulties in handling constraints, which are caused by the fact that algorithms' problem-knowledge and CH strategies might become outdated.

(c) *Some counter-intuitive observations*:

i. The presence of constraints and dynamics in DCOPs might not always make the problems harder to solve. Instead, for certain types of problems, the presence of constraints and dynamics in DCOPs might actually make the problems easier to solve for certain types of algorithms.

ii. Our experiments also show that the presence of constraints always helps algorithms using the repair method like GA+Repair work better.

4. *Suggestion of a list of possible requirements that DO and CH algorithms should meet to solve DCOPs effectively.* This list can be used as a guideline to design new algorithms to solve DCOPs in future research. Details of the list will be summarised in the next subsection.

The research in this chapter also has some limitations, which can be improved in future research. A list of limitations and possible directions to extend this research will be presented in Section 8.2.

5.5.2 Possible requirements for DO and CH algorithms to solve DCOPs effectively

Subsections 5.3.4 and 5.4.3 have suggested two groups of requirements for DO strategies and CH strategies to handle dynamics and constraints effectively in DCOPs. The suggested requirements to handle dynamics can be briefly summarised as follows:

1. If a diversity maintaining mechanism is used, it should be used with an elitism mechanism
2. It might be useful to detect changes in both feasible regions and infeasible regions
3. It might be useful to track the moving feasible regions instead of tracking the moving existing optima.
4. It might be useful to search in both feasible and infeasible regions

The suggested requirements to handle constraints can be briefly summarised as follows:

1. The CH strategies should not affect the way the algorithm handle dynamics. Particularly:

   (a) The CH strategy should allows diversified individuals to be distributed in the whole search space.

   (b) The CH strategy should not reject diversified individuals even if they do not contribute to the CH process.

   (c) Special attention might need to be taken if change-detection is undertaken by monitoring the fitness values of current individual (when there is a drop of individual's performance, we need to check to see if the drop is really caused by an environmental change).

2. The CH strategy needs to get updated whenever a change happens. Particularly:

   (a) The strategy's knowledge about the problem needs to be updated

   (b) The strategy might also need to be updated to deal with new environments

   (c) The strategy should avoid using problem-dependent information because it might not be possible to update this type of information.

In order to solve DCOPs, an algorithm needs to use both DO strategies and CH strategies. As a result, it needs to satisfy both groups of requirements mentioned above. In the next chapter, I will discuss a possible approach of combining DO and CH strategies while still satisfying these two groups requirements. I will then use that approach to develop a set of new algorithms specifically designed to solve DCOPs.

Chapter 6

A new class of algorithms to solve DCOPs

The study in the previous chapter has shown that dynamic constrained optimisation problems (DCOPs) have some special characteristics that make them very different from unconstrained dynamic problems and stationary constrained problems. The aforementioned research also shows that due to these different characteristics, some existing dynamic optimisation (DO) and constraint handling (CH) algorithms might not work effectively in solving DCOPs. The lack of knowledge about DCOPs, the ineffectiveness of existing algorithms in solving continuous DCOPs, and the lack of algorithms specifically designed for solving continuous DCOPs creates an important gap in current dynamic optimisation research.

This chapter contributes to the task of closing this research gap by developing new methods to solve DCOPs more effectively. In this chapter, based on detailed studies in the previous chapter about the common characteristics of DCOPs, the weaknesses of some existing algorithms in solving DCOPs, and our suggested requirements for algorithms to solve DCOPs, I will propose an approach to effectively handle dynamics in DCOPs. The goal is to combine the advantages of DO and CH strategies while overcoming the drawbacks of these methods in solving DCOPs. Specifically, we modify an existing CH technique, the repair method (Michalewicz & Nazhiyath 1995) (Michalewicz n.d.), to create a framework with special mechanisms to support solving DCOPs, and then integrate two DO techniques, random-immigrant (Grefenstette 1992) and hyper-mutation (Cobb 1990), into the framework to develop new algorithms able to solve DCOPs better than the original DO and CH methods.

I will also undertake a detailed analysis to study the behaviours and performance of some existing DO and CH algorithms as well as the newly proposed algorithms in solving DCOPs. Another analysis will also be carried out to investigate more about the characteristics of DCOPs as well as the influence of each algorithmic component on algorithm performance in DCOPs. Some new measures will also be developed to assist these two aforementioned analyses.

The structure of the chapter is as follows. First, in Section 6.1 I will develop a new set of algorithms that are able to overcome the drawbacks of existing DO and CH algorithms. Then detailed experiments and analyses are carried out in Section 6.2 to compare the new algorithms with existing algorithms and to study under what conditions the new algorithms work well. In the next section (Section 6.3), a further analysis will be made to investigate which factors have made the proposed algorithms work well (and why) and whether these factors are the results of our proposed ideas. I will also analyse the proportion of contribution that each of our proposed mechanisms gives in improving algorithms' performance in solving DCOPs. Section 6.4 follows by providing an analysis of the impact of changing parameter values on the performance of the proposed algorithms. Based on this observation I will suggest some recommendations on choosing the suitable parameter values. Finally, in Section 6.5, I will discuss the advantages and disadvantages of the proposed methods and outline future directions.

6.1 A new class of algorithms to solve DCOPs

As DCOPs have the properties of both DO problems and constrained problems, it is natural that in order to solve DCOPs, an algorithm needs to use both DO and CH strategies. In addition, to work effectively in DCOPs the DO and CH strategies chosen by the algorithm need to be modified to satisfy the special requirements outlined in Subsection 5.5.2.

In this section I will firstly discuss the possible DO and CH strategies that we can combine, the possibility to modify them to solve DCOPs effectively, and I will then describe our new algorithms which combines the modified versions of existing DO and CH strategies to solve DCOPs.

6.1.1 Choosing DO and CH strategies

For the experiments in this chapter, I chose the same representative DO and CH methods as used in Chapter 5. They are the DO techniques *triggered hyper-mutation GA* (HyperM (Cobb 1990))

and *random-immigrant GA* (RIGA (Grefenstette 1992)) and the CH technique *repair method* (Michalewicz & Nazhiyath 1995). Because these methods were also used in the previous study of analysing the strengths and weaknesses of DO and CH strategies in solving DCOPs (Chapter 5), using them in this chapter will facilitate us in extending our previous research to gain a deeper understanding of DCOPs.

In order to combine the chosen strategies into a new algorithm to solve DCOPs, we can choose between two approaches. The first approach is to start from DO strategies such as RIGA/HyperM, modify them to create a framework which supports CH strategies and then add CH strategies to the newly developed framework. Alternatively, in the second approach we can start from CH strategies as the repair method, modify it to create a framework which supports DO strategies and then add DO strategies to the framework. In this chapter I will follow the second approach, in which I will modify GA+Repair to support DO strategies and I will then add RIGA/HyperM to the newly created framework.

6.1.2 Potential directions to improve the repair method for solving DCOPs

As shown in the experiments in Subsection 5.4.4 and also in Table 5.10 (page 143), the repair method has some advantages which I believe would make it one of the more suitable CH methods to solve DCOPs. First, the operation of the repair method does not interfere with the operation of such DO strategies as the diversity-maintaining/introducing mechanisms, meaning that the method can be integrated with these DO strategies without too much difficulty. As mentioned earlier in Subsection 5.4.2, some CH strategies may not work well with diversity-maintaining/introducing strategies because these CH strategies select individuals based on their feasibility, i.e. feasible individuals might have a different probability to be selected than infeasible individuals. Such a bias in selection might cause many diversified individuals to be lost because of their infeasibility. The repair method to some extent avoids this drawback because it accepts both feasible and infeasible individuals in the same way, or in other words it does not care about the feasibility of an individual provided that this individual can provide a good (repaired) solution.

Second, the repair method is naturally suitable for tracking the moving feasible due to the way it works. In the repair operation, whenever a search individual is repaired, the newly created repaired individuals will always be closer to existing reference individuals than the search

individual is (see the Repair routine in Algorithm 9, page 151). As a result of that operation, if the algorithm is able to have at least one reference individual in the moving feasible region when changes happen, the repair method will have a chance to send more individuals toward that reference individual and hence will have a chance to track that moving region.

Third, the repair method also naturally supports elitism because the best found feasible solutions will always be stored in the reference population. This property helps the method satisfy one important requirement set out in Subsection 5.5.2 for maintaining diversity effectively in DCOPs.

The aforementioned advantages of the repair method have been empirically confirmed by the experimental results in Subsection 5.4.4 and in (Nguyen & Yao 2010*a*) where although the presence of environmental dynamics does significantly decrease the performance of GA+Repair, the algorithm still has better performance than all other GA-based existing DO algorithms in solving DCOPs. This observation proves that its special characteristics make the repair method a promising approach for solving DCOPs.

However, Subsection 5.5.2 also shows that when solving DCOPs the repair method significantly suffers from the issue of being outdated, a problem that I suspect would affect many existing CH strategies. To apply the repair method to solving DCOPs, we need to improve it to resolve its current drawbacks. Subsection 5.5.2 and Table 5.10 indicate that there are three major requirements that the repair method is not able to satisfy. They are the ability to detect changes, the ability to update its knowledge about the problem and the ability to update the strategy whenever a change happens.

Naturally, a possible direction to improve the performance of GA+Repair in solving DCOPs would be to equip the algorithm with these three features. In the next subsections, I will discuss our proposed method to modify GA+Repair into a framework that supports all three features: detecting changes, updating problem knowledge and updating the CH strategy. I will then discuss the possible way to integrate existing DO strategies into the framework to solve DCOPs more effectively.

6.1.3 Combining the advantages of current DO techniques and CH techniques

Detecting changes

The first improvement that we need to make to the repair method is to develop a change-detection method. Only after being able to detect changes, can we prevent problem information and constraint-handling strategies from being outdated and hence can solve DCOPs better.

Types of changes that need to be detected We need to detect two types of changes that can affect the repair method. The first type includes changes that make the repair method's current knowledge (i.e. the fitness values of search and reference individuals) outdated. Changes of this type occur when the feasibility status or objective values of some special feasible solutions called *influential* feasible solutions change. In repair methods, we call a solution *influential* if its objective value has been used to calculate the fitness values of search individuals or reference individuals in the population and hence there is a mapping between that solution and the corresponding individuals. For example, in step 3c of Algorithm 9 (page 151), $\mathbf{z}$ is the influential solution of the search individual $\mathbf{s}$ because $\text{eval}(\mathbf{s}) = f(\mathbf{z})$. Similarly, in step 3b of Algorithm 9, $\mathbf{z}$ is also the influential solution of the reference individual $\mathbf{r}$ because $\text{eval}(\mathbf{r}) = f(\mathbf{z})$. If a change affects such an influential solution by changing its objective value or its feasibility, the existing mapping between the solution and its corresponding individuals might no longer reflect the new objective value or feasibility of the solution and consequently the repair method might become outdated. The better the objective value of an influential solution is, the more influence it has and we should at least detect the changes occurring in the most influential solutions. It is of note that, in the repair method the most influential solutions, i.e. the ones with the best objective values, are always retained as members of the reference population. For example, in step 3b of Algorithm 9, page 151, the influential solution $\mathbf{z}$ is retained as a reference individual ($\mathbf{r} = \mathbf{z}$). Due to that, in order to detect changes in the most influential solutions we *only* need to monitor the changes in objective values and feasibility of reference individuals.

The second type of changes that need to be detected includes changes that occur in areas not covered by the algorithm's population. As these areas are not covered, in case a change happens it would go unnoticed by the algorithm for a certain period. Changes of this type can occur in two different forms: (a) changes in objective functions and (b) changes in feasibility

(constraints). In case (a), changes in objective functions will only affect the repair method if they are not covered by the current population **and** occur in *feasible regions* (for objective changes in an infeasible region, all solutions in the region still remain infeasible and hence the change would not affect the repair method). We can use existing unconstrained DO change detection techniques to monitor these objective changes. In case (b), changes in constraints will only affect the repair method if they *alter* the shape of feasible/infeasible regions so that the current global optimum become infeasible or inversely an infeasible solution become the new global optimum. The first case is caused by the extension of existing infeasible regions or the occurrence of a new infeasible region, and the second case is caused by the shrink of existing infeasible regions or the occurrence of a new feasible region. Due to that, in order to detect constraint changes of this type we only need to monitor the shrink/extension/appearance of infeasible/feasible regions.

The proposed change-detection mechanisms To detect the two different types of changes above, we need to use different change-detection methods. For changes in objective function, we can adopt and modify the change-detection method originally used in HyperM (Cobb 1990): monitor the drop of the average best fitness values over a certain period. Unlike the original change-detection method in HyperM, here we will only monitor the drop of the average fitness values of the best *influential* individuals, which are also the best individuals in the reference population. If we detect a drop in the average best reference fitness values and the drop persists over a certain period, we can assume that a change in the objective function has occurred and that change might make the repair method outdated. Details of the procedure to detect changes in objective function can be found in the prosecode of routine DetectChange() (Algorithm 10, page 167).

For changes in constraints, we developed a new change-detection method. As discussed above, because the only types of constraint changes that might affect the algorithms are the extension, shrink, or appearance of infeasible/feasible areas, we can detect these types of changes by monitoring the feasibility/infeasibility of individuals that are near the infeasible boundaries. We also need to keep checking the feasibility/infeasibility of the best individual in the reference population because it is the current most influential solution.

To detect the shrink of infeasible areas, at each generation I chose to monitor some *infeasible individuals* that are near the feasible boundaries, i.e. those that satisfy the following conditions:

Algorithm 10 *routine* DetectChange()

Notes	The routine needs to be called at each generation
Inputs	k - index of the current generation
	b^k - the fitness value of best feasible solution at gen. k
	X_u^{k-1} - set of up to q *unfeasible* individuals that have smallest sum $\sum g_i(\mathbf{x})$ at generation $k-1$
	$\overline{X}_u^{k-1}$ - set of up to $q \operatorname{div} 2$ most diversified *unfeasible* individuals in X_u^k at generation $k-1$
	$\overline{X}_f^{k-1}$ - set of q *feasible* individuals that are closest to individuals in $\overline{X}_u^k$ at generation $k-1$
Outputs	**returnValue** – whether a change is detected or not
	$\overline{X}_u^k$ and $\overline{X}_f^k$ - for use in the next generation

1. *Detect changes near the boundaries of infeasible regions:* For each $\mathbf{x}_i \in \overline{X}_u^{k-1}$,
   (a) Re-evaluate the constraint functions $g(\mathbf{x}_i)$
   (b) **If** $\mathbf{x}_i$ becomes *feasible*,**returnValue=true**,go to step 6
2. *Detect changes near the boundaries of feasible regions:* For each $\mathbf{x}_i \in \overline{X}_f^{k-1}$,
   (a) Re-evaluate the constraint functions $g(\mathbf{x}_i)$
   (b) **If** $\mathbf{x}_i$ becomes *unfeasible*,**returnValue=true**,go to step 6
3. *Detect the performance drop of the best feasible solution*
   (a) if $\sum_{k-4}^{k}(b^i/5)$ is worse than $\sum_{k-5}^{k-1}(b^i/5)$ **returnValue=true**, go to step 6
4. *Detect feasibility change of best feasible solution*: if it becomes infeasible, **returnValue=true**, go to step 6
5. *If nothing detected*, **returnValue=false**, go to step 6
6. *Update and return*:
   (a) Remove $\overline{X}_u^{k-1}$and $\overline{X}_f^{k-1}$from the memory
   (b) $X_u^k = \overline{X}_u^k = \overline{X}_f^k = \{\emptyset\}$
   (c) Initialise X_u^k: Add up to q *unfeasible* individuals that have the smallest sum $\sum g_i(\mathbf{x})$to X_u^k
   (d) Initialise $\overline{X}_u^k$: Move up to $q \operatorname{div} 2$ most diversified individuals from X_u^k to $\overline{X}_u^k$: **For** $i : 1$ **to** $q \operatorname{div} 2$
      i. Select $\mathbf{x}_i \in X_u^k$ that has the maximum total distances to all individuals in $\overline{X}_u^k$
      ii. Remove $\mathbf{x}_i$ from X_u^k and add $\mathbf{x}_i$ to $\overline{X}_u^k$
   (e) Initialise $\overline{X}_f^k$: Add up to $q \operatorname{div} 2$ *feasible* individuals, which are closest to the individuals in $\overline{X}_u^k$, to $\overline{X}_f^k$
   (f) Add $\overline{X}_u^k$and $\overline{X}_f^k$ to system's memory
   (g) **Return returnValue;**

- q infeasible individuals $\mathbf{x}_j$ that have the smallest sum $\sum_i g_i(\mathbf{x}_j)$ where g_i are the constraint functions; $j = 1 : q$ (for an individual $\mathbf{x}_j$, the smaller the sum $\sum_i g_i(\mathbf{x}_j)$, the more likely that $\mathbf{x}_j$ is close to the boundaries of the feasible regions)

- Among these individuals, select $q \operatorname{div} 2$ most diversified individuals, which are those that have the farthest distance to each other. This is to make sure that we are monitoring different boundaries

To detect the extension of infeasible areas, in each generation I also chose to monitor $q \operatorname{div} 2$ *feasible individuals* that are closest to the $q \operatorname{div} 2$ *infeasible individuals* chosen above.

The $q \operatorname{div} 2$ infeasible individuals and $q \operatorname{div} 2$ feasible individuals chosen above, along with their current feasibility status, will then be stored in a temporary memory *for one generation*. At the next generation, the constraint functions of these $2 \times (q \operatorname{div} 2)$ chosen individuals are re-evaluated to see if they still have the same feasible/infeasible status. If any feasible individual becomes infeasible and vice versa, we can assume that there is a change in the constraints and this change might make the repair method outdated. After this feasibility-reevaluation process finishes, the chosen individuals will be removed from the temporary memory. Details of the procedure to detect changes in objective function can be found in the prosecode of routine DetectChange() (Algorithm 10, page 167).

For this change-detection method, there is one parameter that we need to take into account. This is q, the number of feasibility detectors used to detect constraint changes. The value of q determines the number of constraint function evaluations to be made at each generation. Although it is generally assumed that the cost of constraint function evaluations is not as significant as the cost of objective function evaluations, if the constraint functions are computationally expensive or if there are many constraint functions, a large value of q might affect the performance.

Due to this fact, an adaptation mechanism should be used to determine the value of q depending on the size of the population, the number of constraint functions and the scale of the objective function (possibly represented by its dimensionality). If the population-size, the number-of-constraints, and the dimensionality are relatively small, then the value of q can be close to the population size. Otherwise, q should only be equal to a fraction of the population size. Following this guideline, in this chapter I will propose a semi-adaptive mechanism to calculate the value of q depending on the population size, the number of constraint functions and the number

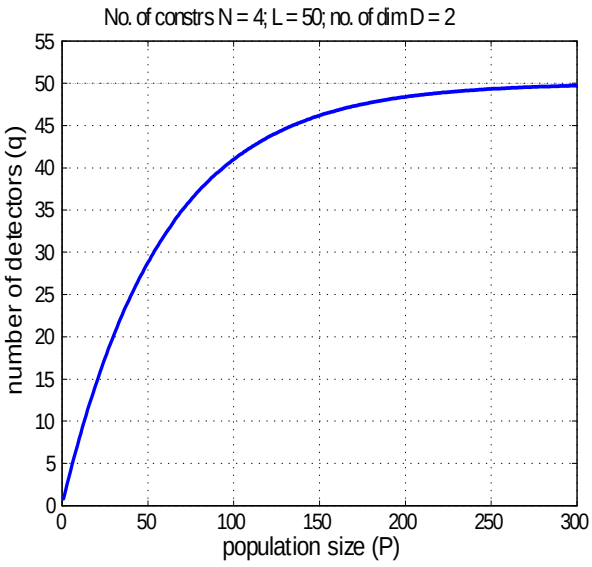

Figure 6.1: This figure shows how the feasibility-change-detection method would select the appropriate number of feasibility detectors to maintain an efficient computational cost given the population size P, the number of constraint functions N, the number of dimensions D and the allowable limit of detectors L (in this graph the latter three are fixed for the purpose of illustration). As can be seen in the figure, when the population size is small, most individuals can be used as detectors. When the population size becomes larger, the number of detectors also become larger until it reaches the upper-limit but the proportion between number of detectors and population size and gradually becomes smaller to save computational costs.

of dimensions (variables). The mechanism is described in equation (6.1) and a graph showing the value of q calculated based on different population sizes/number-of-constraints/number-of-dimensions is given in Figure 6.1. The following equation is used to calculate the value of q

$$q = \min\left(P, \left(1 - \exp\left(-\alpha P\sqrt{N+D}\right)\right) L\right) \tag{6.1}$$

where P is the population size, N is the number of constraint functions, D is the number of dimensions, L is the maximum number of detectors allowed by users (in this chapter $L = 50$), and $\alpha = 0.007$ is a constant. α is used to control the steepness of the "curve" of q in Figure 6.1 when one or all the value of P, N and D increase. The larger the value of α, the steeper the curve.

Advantages and disadvantages of the change-detection mechanism The first and most obvious advantage of the newly proposed change-detection mechanism is that it helps the repair method to react to environmental changes and to update any outdated information/strategy

promptly. This consequently would help the repair method to satisfy a number of requirements set out in Subsection 5.5.2.

Second, the new change-detection mechanism also does not require any additional objective function evaluation except at most one evaluation per generation to re-evaluate the best individual in case it has not been evaluated in the repair process.

Third, although the change-detection mechanism requires a number of constraint function evaluations to be taken, it makes sure that this number of evaluations is not too high and is in proportion with the scale of the problem and with the size of the population.

Fourth, similar to other diversity-introducing methods, to some extent the change-detection mechanism might help to save some computational cost dedicated to dealing with changes. When being used with this mechanism, only until the time when a change is detected, do the algorithms need to spend additional computational cost to react to the change.

It should be noted that because this change-detection mechanism is designed to work with the repair method only and to detect only changes that make the repair method outdated, the method might not be able to detect all other types of changes. As this proposed change-detection method only select detectors from the population, if the algorithm converges on a certain area, the algorithm might not be able to detect all other types of changes in other areas of the landscape.

This disadvantage, however, can be improved by hybridising the newly proposed change-detection mechanism with some diversity-maintaining mechanism to make the population more diversified. That way it might be easier to detect changes that occur in other parts of the landscape and thanks to that the new change-detection mechanism can be applied effectively to other dynamic optimisation algorithms. As we will see later it turns out that actually GA+repair itself already has a good diversity level and hence it is able to partly alleviate this disadvantage. In Subsection 6.1.3 I will also discuss the possibility of hybridising GA+Repair with such diversity-maintaining/introducing mechanisms as RIGA and HyperM to effectively overcome the disadvantage.

Another disadvantage of the change-detection mechanism is the additional computational cost. For each constraint function, at each generation the change-detection mechanism needs to perform additional $2\times$ $(q \operatorname{div} 2)$ constraint evaluations. Although the adaptive mechanism in Equation 6.1 has been proposed to makes sure that this number of evaluations is not too high and

is in proportion with the scale of the problem and with the size of the population, the cost might still negatively affect algorithm performance if the constraint functions are computationally expensive.

Updating reference individuals

As been analysed in Subsection 5.4.4 and also in (Nguyen & Yao 2010*a*), one of the reasons for the inefficiency of GA+Repair and possibly other repair methods in solving DCOPs is that the reference individuals might become outdated when a change happens. In order to resolve this issue, we need to update those reference individuals that have become outdated so that they can correctly reflect the new landscape after a change.

In this chapter, I propose a simple method to update the outdated reference individuals. First, for each infeasible reference individual, we will try to replace it by a feasible individual from the search population using non-linear ranking selection. If there is no feasible individual in the search population, we will replace the infeasible reference individual with a feasible individual from the reference population. If there is no such feasible individual, we will then replace the infeasible reference individual with a randomly generated feasible solution. After all infeasible reference individuals have been replaced by the feasible ones, we will then re-evaluate all reference individuals if they have not been evaluated since the last change.

The update procedure for the reference population is described in the prosecode in Algorithm 11 (page 172).

Updating search individuals

As also been analysed in Subsection 5.4.4, another reason for the inefficiency of GA+Repair and possibly other repair methods in solving DCOPs is that some search individuals might also become outdated when a change happens if they are not selected for the repair process.

In order to resolve this issue, we need to update those search individuals that have become outdated so that they can correctly reflect the new landscape after change.

The update mechanism that I use in this chapter is very simple. Whenever we detect a change that affects the repair method, I firstly update the reference population and then use the newly updated reference individuals to repair all search individuals that have not been repaired since the last change. This is to make sure that the fitness values of all search individuals correctly reflect the feasible regions of the newly changed landscape. The update procedure is

Algorithm 11 *routine* UpdateReferencePop()

Variables: S - Search population
R - Reference population
n_S - Number of feasible individuals in S
n_R - Number of feasible individuals in R
m_R - Number of unfeasible individuals in R

1. **For** $i = 1$ **to** $\min(n_S, m_R)$: Replace each unfeasible individual x_u^i in R with a feasible one from S
   (a) Use non-linear ranking selection to choose a feasible individual $x_f$from S
   (b) Replace $x_u^i$by x_f: $x_u^i = x_f$
2. **If** $(m_R > n_S)$**then For** $j = n_S + 1$ **to** $n_S + \min(m_R - n_S, n_R)$: Replace each unfeasible individual x_u^j in R with a feasible one from R
   (a) Use non-linear ranking selection to choose a feasible individual $x_f$from R
   (b) Replace $x_u^j$with x_f: $x_u^j = x_f$
3. **If** $(m_R > n_S + n_R)$**then For** $k = n_S + n_R + 1$ **to** $m_R - n_S - n_R$: Replace each unfeasible individual x_u^k in R with a feasible, randomly generated individual
4. Re-evaluate all reference individuals that have not been evaluated since the last change

described in the prosecode in Algorithm 12 (page 172).

Algorithm 12 *routine* UpdateSearchPop()

Variables: S - Search population
Note: This routine needs to be called after the routine UpdateReferencePop (Algorithm 11, page 172)

1. Create a set $S_1 \in S$ which includes all search individuals that have not been selected for the repair process since the last change
2. For each individual s in S_1, update the fitness value of s by calling the routine *Repair* (**s**)

Maintaining/introducing diversity

Numerous previous studies have shown that maintaining/introducing diversity are necessary for DO. In addition, as discussed earlier, maintaining a high level of diversity in the population so that the algorithm is able to cover a large area or ideally the whole landscape would help the change-detection methods proposed in Subsection 6.1.3 to work better.

To enhance diversity in the repair method, in this chapter I also hybridise the mutation strategies of RIGA and HyperM with GA+Repair. The implementation is very simple: we just

replace the normal GA mutation with the mutation strategies of RIGA and HyperM. Details of the implementation will be described later in subsection 6.1.4.

Searching out of range

From our experiments I have found that the original repair method has another drawback: it becomes less effective in solving problems with the global optimum in boundaries of the search region. The reason for this is because the probability of the repair method finding the optimum is smaller when the optimum is in the boundaries of the search region than when it is not.

One way to make repair methods work better in problems with optima in boundaries of the search region is to allow the algorithm to search out of range and consider all out-of-range solutions infeasible. In an extended search space like that, the actual global optimum now is inside the search area, making it easier for the repair method to find the optimum. When extending the search space like this, one question is how much should we extend the search space to keep the algorithms working effectively. In this chapter, when the out-of-range mechanism is tested we allow the algorithms to search 25% beyond the given search range.

6.1.4 The dRepairGA algorithm and other variants

In the previous subsections, I have proposed a set of different mechanisms which can be used as a framework to improve the performance of the repair method in solving DCOPs. In this subsection, I will apply these mechanisms to an algorithm, the GA+Repair, to evaluate how effective the proposed mechanisms would be in solving DCOPs. The mechanisms are combined with GA+Repair to create three different versions of new algorithms.

For the first version of the new algorithms, we integrate the change-detection and update mechanisms to GA+Repair to create a new algorithm called dRepairGA. The purpose is to see how the change-detection and update methods would help the repair method to cope with DCOPs. The integration in dRepairGA is simple. In addition to all previous operations of GA+Repair, at every generation we call the routine `DetectChange()` to detect any environmental changes that may possibly affect the repair method. If a change is detected, we will invoke the routine `UpdateReferencePop()`, followed by the routine `UpdateSearchPop()`, to update the search population and the reference population to deal with the change. The prosecode of the algorithm is described in Algorithm 13, page 175.

To investigate if the proposed mechanisms can be applied to other algorithms, I also de-

velop an improved version of Genocop III by hybridising the original Genocop III algorithms with our newly proposed mechanisms (routines `DetectChange()`,`UpdateReferencePop()`and `UpdateSearchPop()`) to handle dynamics and dynamic constraints. These routines are integrated into Genocop in the same way as we do in dRepairGA in Algorithm 13, page 175. The new algorithm is called dGenocop.

dRepairGA was then extended to a second version. In this second version, we hybridised existing DO strategies such as RIGA and HyperM with dRepairGA. The purpose is to investigate the usefulness of combining existing DO strategies with our CH strategies, and to see if our proposed mechanisms can help to avoid the previously known drawbacks of RIGA and HyperM in solving DCOPs. The new algorithms are called dRepairRIGA and dRepairHyperM.

These two algorithms are almost identical to dRepairGA except that they have the mutation strategy of RIGA and HyperM instead of the basic GA's mutation strategy. Specifically, in dRepairRIGA, in addition to the normal GA mutation rate a fraction of the population (represented by the *random-immigration rate*) is replaced by random solutions at every generation. In dRepairHyperM, the basic mutation rate of GA is still kept, but when a change is detected that mutation rate will be replaced by the *hyper-mutation rate* of HyperM. The algorithm will keep using the hyper-mutation rate until no performance drop is recognised. In that case the algorithm will resume back to its normal base-mutation rate. It should be noted that there is a difference in the mutation strategy of dRepairHyperM and HyperM. This is the fact that dRepairHyperM can trigger its hyper-mutation not only when it detects a drop in performance of the best individual like HyperM, but also when the DetectChange() routine returns true.

It is also worth noting that such repair-based algorithms as dRepairRIGA and dRepairHyperM always have a lower level of diversity than RIGA and HyperM given the same mutation/random-immigrant rate. This can be attributed to the fact that, in repair-based algorithms the search population and the reference population have different evolving period. While the search population evolves at every generation, the reference population only evolves after each 100 evaluations. Due to that, during each 100-evaluation period no mutation is applied to the reference individuals and consequently there is less diversity in repair-based methods given the same mutation/replacement rate as GA-based algorithms. In addition, when the reference population evolves, individuals generated by mutation or random-immigrant are only accepted if they are feasible and better than their parents. With our current implementation, the number of in-

dividuals that are mutated/replaced in dRepairHyperM/dRepairRIGA is only equal to about less than 80% of those in RIGA/HyperM with the same hyper-mutation and random-immigrant rate. In other words, a higher mutation rate of 0.8 in dRepairHyperM/dRepairRIGA would only generate the same number of diversified individuals as a lower mutation rate of less than 0.64 in RIGA/HyperM.

In the third version of the algorithms, because I observed that the repair operator does not work very well in problems with optima in boundaries of the search region, I also implement a version of dRepairGA/RIGA/HyperM which can search 25% beyond the given search range using the out-of-range (OOR) mechanism proposed previously in Subsection 6.1.3. All out-of-range solutions are considered infeasible. The algorithms with the out-of-range search mechanism are called dRepairGA_OOR, dRepairRIGA_OOR and dRepairHyperM_OOR.

Algorithm 13 dRepairGA

Note: It is assumed that the problem is maximisation
Routines: `DetectChange`() - described in Algorithm 10, page 167
`UpdateReferencePop`() - described in Algorithm 11, page 172
`UpdateSearchPop`() - described in Algorithm 12, page 172

1. *Initialise*: same as step 1 of GA+Repair (Algorithm 8,page 150)
2. *Search at each generation*: same as step 2 of GA+Repair
3. *Evolve the reference population*: same as step 3 of GA+Repair
4. *Detect change and update search strategy*:
   (a) For each generation: **If** `DetectChange`()=**true**
      i. Update reference pop.: `UpdateReferencePop`()
      ii. Update the search pop.: `UpdateSearchPop`()
   (b) **Else** do nothing
5. *Return to step 2*

6.1.5 Related research in the continuous domain

Only until very recently a few algorithms specially designed for DCOPs were proposed. In (Nguyen & Yao 2009*a*), we made the first attempt to develop a GA-based algorithm named RepairGA to solve DCOPs. The algorithm is also based on the repair method. It is a combination of the algorithm GA+Repair mentioned in Subsection 5.4.4 and a simple mechanism to update reference individuals at every generation. Experimental results in (Nguyen & Yao 2009*a*) show

that RepairGA performs better than existing DO algorithms such as GA/RIGA/HyperM in the four tested problems. Compared to the new dRepairGA and its variants proposed in this section, the repair-based algorithm in (Nguyen & Yao 2009*a*) is significantly different and less efficient because it lacks the following mechanisms to handle DCOPs: (1) an explicit mechanism to adaptively detect changes; (2) a mechanism to update outdated search individuals; (3) a mechanism to maintain/introduce diversity in dynamic environments; and (4) a mechanism to search out of range to find optima in boundaries of the search region more effectively. The newly proposed dRepairGA and its variants can be seen as extensions of the old RepairGA algorithm where the four mechanisms above are incorporated to solve DCOPs better. Our experimental results (not shown) also show that dRepairGA perform significantly better than RepairGA in solving DCOPs.

Using two of the benchmark problems proposed in (Nguyen & Yao 2009*a*), in (Singh *et al.* 2009) a static constraint optimisation algorithm (IDEA) was evaluated in DCOPs. IDEA is an EA that uses a special ranking mechanism to select individuals: the algorithm explicitly maintains some infeasible individuals during the search and ranks "good' infeasible solutions higher than feasible solutions. This algorithm, however, was not actually designed for solving dynamic environments but static environments. In (Singh *et al.* 2009), to make it work in DCOPs IDEA was modified by adding a simple change-detection method in which at each generation a random individual is chosen to detect changes. Whenever a change is detected, the whole population is re-evaluated to make sure that the information that the algorithm has is updated. The simple change-detection method used in IDEA does not guarantee that changes are always detectable because it assumes that changes can be detectable by re-evaluating any random individual in the search space, which is not always the case. The IDEA algorithm is different from the algorithms proposed in this chapter because (1) it uses a different method to handle constraints; (2) it does not use any DO techniques except change-detection to handle changes; and (3) its change-detection is simple and possibly insufficient in certain cases. Experimental results show that IDEA perform betters than the chosen peer EA in the two tested problems.

The latest algorithm that was proposed specifically to solve DCOPs is the study of Richter (Richter 2010), which was published at the time when this chapter was being prepared for submission. In this study, a special memory-scheme, abstract memory, was adapted for DCOPs. Abstract memory is a special memory scheme which relies on a probabilistic model of the oc-

currence of good solutions in the search space to memorise a "spatio–temporal cartographing" (Richter & Yang 2009) of promising regions in the search space. In (Richter 2010), this scheme was adapted for DCOPs by separately memorising the good candidates for solving the unconstrained objective function as well as the likely feasible regions. Elements from the two memories then are processed using two different schemes: *blending* and *censoring*. The memory schemes were then applied to an EA using a penalty function and an EA using the repair method to solve one benchmark problem. The result show that the proposed memory schemes can help improving the performance of the tested EA in certain situations. The blending and censoring memory-based EAs proposed in (Richter 2010) are very different from the new algorithms proposed in this chapter, especially in the following major points: (1) they follow a different approach (using memory-based mechanisms); (2) to detect changes, they only use the HyperM's fitness-drop monitoring mechanism, which might not be effective in the case of newly-appearing-optima DCOPs.

6.2 Comparing and analysing dRepairGA and its variants against existing algorithms

6.2.1 Chosen algorithms

In this section the performance of different versions of dRepairGA and dGenocop (as described in the previous section) will be compared with those of existing algorithms: GA, RIGA, HyperM, GA+Repair and Genocop III. The purpose is to see if our newly proposed mechanisms can help improving the drawbacks of existing methods. As these algorithms are all based on basic GA and the only difference between them are the additional mutation strategy / change-detection strategy that they use to handle dynamics, by comparing these algorithms we will be able to identify if the strategies they employ are effective in solving DCOPs.

It should also be noted that dRepairGA/dRepairRIGA/dRepairHyperM and their out-of-range enabling versions will be compared with the original GA/RIGA/HyperM/GA+Repair and dGenocop will be compared with the original Genocop III. The reason why I do not compare the performance of dRepairGA-based algorithms with that of Genocop III is that it is not easy to see if the better or worse performance of dRepairGAs compared to Genocop III is due to our proposed strategy or not. This is due to that Genocop III is very different from GA as already

Table 6.1: Test settings for all algorithms used in the paper

All algorithms (exceptions below)	Pop size	25
	Elitism	Elitism & non-elitism if applicable
	Selection method	Non-linear ranking as in (Michalewicz n.d.)
	Mutation method	Uniform, $P = 0.15$
	Crossover method	Arithmetic, $P = 0.1$
HyperM & variants	Triggered mutate	Uniform, $P = 0.5$ as in (Cobb 1990)
RIGA & variants	Rand-immig. rate	$P = 0.3$ as in (Grefenstette 1992)
GA+Repair, dRepairGA & variants	Search pop size	20
	Reference pop size	5
	Replacement rate	0 (default is 0.25 as in (Michalewicz n.d.))
Genocop & variants	Search pop size	20
	Reference pop size	5
	Other parameters	Default as in (Michalewicz n.d.)
Benchmark problem settings	Number of runs	50
	Number of changes	10
	Change frequency	1000 objective-function evaluations
	ObjFunc severity k	0.5 (medium), except G24_6a/b/c/d where $k = 1$ (large severity)
	Constr. severity S	20 (medium)

explained in paragraph 4, Subsection 6.1.1.

6.2.2 Parameter settings

Table 6.1 (page 178) shows the detailed parameter settings for all algorithms tested in this chapter. To create a fair testing environment, all algorithms, including the newly proposed algorithms, use the same parameter values as in the previous experiments in Subsections 5.3.3 and 5.4.4. Existing DO algorithms (GA/RIGA/HyperM) also use the same penalty methods as described in Subsection 5.3.3.

The algorithms were tested in 18 benchmark problems described in Section 5.2 at a change-severity level of medium, except in G24_6a/b/c/d where the severity level is always high (high severity is a property of these four problems).

6.2.3 Performance measures and analysis criteria

To measure the performance of the algorithms in this particular experiment, I use the *modified offline error for DCOPs* proposed in Subsection 5.3.2 (Eq. 5.3).

The full *offline error* scores of all algorithms in the test can be seen in Table 6.2 (page 181). The data in this table is provided mainly for reference purpose only because similar to

the experiments in the previous chapter, to achieve a better understanding of how well the newly proposed algorithms work in different types of problems and how each characteristic of DCOPs would affect the performance of the new algorithms, we further analyse the results by studying them from different perspectives. First, I summarise the average performance of the tested algorithms in each major group of problems (see test results in Figure 6.2, page 182). Then, I investigate the effect of each problem characteristic on each algorithm by analysing their performance in 21 test cases (pair of almost identical problems, one with a particular characteristic and one without, as shown in Table 5.5, page 108). The results of this pair-wise analysis are shown in Figure 6.3, page 183 and Figure 6.4, page 184.

It should also be noted that, in the aforementioned Table 6.2, readers might notice that we included not only the already-described algorithms such as GA, RIGA, HyperM, GA+Repair, dRepairGA, dRepairRIGA, dRepairHyperM, dRepairGA_OOR, dRepairRIGA_OOR, dRepairHyperM_OOR, Genocop and dGenocop but also some other algorithms which have not been introduced yet. These algorithms will be introduced and analysed in the later sections. For now, in this section I will only focus on the data relating to the algorithms that I have described previously.

In the following subsections, I will analyse the performance of dRepairGA-based algorithms against existing algorithms in different classes of problems using the performance measure and criteria above. I will test the algorithms in not only DCOPs, but also static (constrained and unconstrained) problems and unconstrained dynamic problems. The purpose is to see how robust each algorithm is in solving different types of problems. For each class of problems, I will first compare the performance of dRepairGA-based algorithms with existing DO algorithms (GA/RIGA/HyperM, both elitism (-elit) and non-elitism (-noElit) versions) and then I will compare the performance of dRepairGA-based algorithms with existing CH algorithms (GA+Repair/Genocop).

Overall performance

Overall, 6.2 shows that with a precision level of three significant digits, the newly proposed dGenocop is the overall best algorithm. It achieves the top results in 9 out 18 problems, followed by the newly proposed GenocopwUPCwNRR, which achieves the top results in 5/18 problems. Our modified versions of HyperM: dRepairHyperM and its out-of-range dRepairHyperM-OOR

also work well: each algorithm achieve top results in 3 problems, followed by dRepairRIGA/dRepairRIGA-OOR (each has 2/18 best results) and dRepairGA (1/18 best result). It is interesting to note that the original Genocop also work really well, achieving the top results in two static constrained problems and one dynamic constrained problem. This observation and the fact that dGenocop perform better than dRepairGA suggest that the other existing contraint-handling operators in Genocop are also very useful in handling dynamic constrained problems.

In the next subsections we will analyse in details the performance of each algorithm in different group of problems.

Table 6.2: Averaged modified offline errors of all tested algorithms in all 18 problems after 50 runs.

	G24-u (dF,noC)		G24-1 (dF, fC)		G24-f (fF, fC)		G24-uf (fF, noC)		G24-2 (dF, fC)		G24-2u (dF,noC)		G24-3 (fF,dC)		G24-3b (dF,dC)		G24-3f (fF, fC)	
Algorithm	mean	stdDev	mean	stdDev	mean	stdDev	mean	stdDev	mean	stdDev	mean	stdDev	mean	stdDev	mean	stdDev	mean	stdDev
.GA-noElit	0.361	0.064	0.803	0.041	0.802	0.065	0.546	0.033	0.429	0.061	0.206	0.021	0.884	0.093	0.997	0.095	1.143	0.225
.RIGA-noElit	0.234	0.019	0.633	0.046	0.668	0.079	0.449	0.049	0.351	0.060	0.139	0.032	0.742	0.100	0.849	0.093	0.791	0.085
.HyperM-noElit	0.249	0.034	0.450	0.094	0.219	0.085	0.148	0.051	0.304	0.025	0.144	0.026	0.692	0.094	0.506	0.018	0.248	0.071
.GA-elit	0.214	0.037	0.587	0.085	0.227	0.065	0.095	0.044	0.329	0.074	0.103	0.022	0.384	0.092	0.637	0.101	0.241	0.051
.RIGA-elit	0.131	0.034	0.401	0.046	0.171	0.100	0.086	0.028	0.283	0.021	0.110	0.030	0.340	0.045	0.472	0.053	0.203	0.042
.HyperM-elit	0.173	0.042	0.475	0.060	0.224	0.052	0.093	0.032	0.376	0.055	0.111	0.030	0.561	0.104	0.511	0.115	0.142	0.058
.GA+Repair	0.468	0.059	0.226	0.024	0.041	0.011	0.218	0.018	0.281	0.036	0.294	0.029	0.156	0.008	0.171	0.019	0.025	0.008
.GA+RepairwUPGwNRR	0.602	0.211	0.624	0.202	0.238	0.159	0.813	0.224	0.497	0.066	0.573	0.092	0.239	0.128	0.421	0.142	0.040	0.047
.GA+RepairwUPGwRR	0.577	0.082	0.620	0.153	0.238	0.137	0.807	0.226	0.455	0.109	0.550	0.128	0.200	0.098	0.397	0.105	0.038	0.008
.GA+RepairwUPCwNRR	0.306	0.084	0.104	0.025	0.041	0.009	0.218	0.030	0.202	0.027	0.198	0.015	0.038	0.007	0.075	0.013	0.025	0.004
.dRepairGA	0.362	0.063	0.101	0.022	0.042	0.011	0.219	0.073	0.198	0.030	0.201	0.023	**0.034**	0.005	0.079	0.012	0.025	0.002
.dRepairRIGA	0.254	0.048	**0.082**	0.015	0.028	0.006	0.194	0.043	0.162	0.021	0.187	0.011	**0.029**	0.004	0.058	0.007	0.014	0.002
.dRepairHyperM	0.319	0.034	0.093	0.023	0.045	0.010	0.218	0.050	0.171	0.026	0.196	0.024	**0.027**	0.005	0.071	0.014	0.025	0.005
.dRepairGA-OOR	0.156	0.018	0.104	0.025	0.041	0.010	0.248	0.080	0.196	0.035	0.084	0.030	**0.035**	0.007	0.075	0.012	0.026	0.004
.dRepairRIGA-OOR	0.152	0.026	**0.078**	0.014	0.029	0.009	0.151	0.032	0.171	0.021	0.082	0.010	**0.029**	0.005	0.059	0.013	0.014	0.002
.dRepairHyperM-OOR	0.175	0.040	0.091	0.016	0.043	0.010	0.249	0.072	0.161	0.029	0.096	0.022	**0.026**	0.005	0.074	0.014	0.027	0.006
.Genocop	0.120	0.028	0.099	0.034	**0.020**	0.008	**0.030**	0.008	0.177	0.031	0.120	0.028	0.099	0.034	0.020	0.008	0.030	0.008
.GenocopwUPGwNRR	0.302	0.101	0.494	0.167	0.109	0.072	0.170	0.057	0.701	0.100	0.302	0.101	0.494	0.167	0.109	0.072	0.170	0.057
.GenocopwUPGwRR	0.412	0.195	0.719	0.185	0.107	0.083	0.170	0.053	0.638	0.178	0.412	0.195	0.719	0.185	0.107	0.083	0.170	0.053
.GenocopwUPCwNRR	0.123	0.029	0.103	0.024	**0.022**	0.014	**0.029**	0.016	0.138	0.030	0.123	0.029	0.103	0.024	**0.022**	0.014	0.029	0.016
.dGenocop	**0.091**	0.035	0.085	0.024	**0.021**	0.014	**0.030**	0.030	**0.099**	0.028	**0.060**	0.033	**0.028**	0.007	0.068	0.022	**0.005**	0.003

	G24-4 (dF, dC)		G24-5 (dF,dC)		G24-6a(2DR,hard)		G24-6b (1R)		G24-6c (2DR,easy)		G24-6d (2DR,hard)		G24-7 (fF, dC)		G24-8a(nC,ONISB)		G24-8b (fC,OICB)	
Algorithm	mean	stdDev	mean	stdDev	mean	stdDev	mean	stdDev	mean	stdDev	mean	stdDev	mean	stdDev	mean	stdDev	mean	stdDev
.GA-noElit	0.718	0.095	0.465	0.061	0.630	0.085	0.541	0.043	0.575	0.057	0.730	0.077	0.913	0.134	0.479	0.054	1.327	0.111
.RIGA-noElit	0.612	0.052	0.401	0.048	0.455	0.065	0.380	0.051	0.412	0.051	0.461	0.050	0.713	0.022	**0.392**	0.040	1.216	0.093
.HyperM-noElit	0.562	0.062	0.347	0.068	0.437	0.067	0.439	0.062	0.402	0.050	0.481	0.042	0.544	0.079	**0.392**	0.019	1.178	0.099
.GA-elit	0.627	0.045	0.373	0.031	0.826	0.154	0.571	0.071	0.563	0.062	0.614	0.108	0.518	0.095	0.409	0.027	1.303	0.083
.RIGA-elit	0.492	0.071	0.259	0.031	0.458	0.050	0.426	0.045	0.413	0.040	0.427	0.028	0.459	0.057	0.410	0.019	1.085	0.111
.HyperM-elit	0.494	0.039	0.297	0.047	0.575	0.074	0.469	0.074	0.527	0.039	0.585	0.057	0.478	0.072	0.406	0.046	1.081	0.084
.GA+Repair	0.211	0.015	0.236	0.024	0.431	0.074	0.427	0.048	0.390	0.038	0.354	0.038	0.181	0.017	0.496	0.032	0.391	0.068
.GA+RepairwUPGwNRR	0.339	0.137	0.286	0.059	0.744	0.439	0.708	0.185	0.620	0.136	0.516	0.208	0.351	0.163	0.448	0.100	1.327	0.116
.GA+RepairwUPGwRR	0.356	0.101	0.281	0.084	0.454	0.154	0.656	0.151	0.613	0.103	0.588	0.173	0.312	0.136	0.415	0.092	1.149	0.209
.GA+RepairwUPCwNRR	0.178	0.018	0.181	0.023	0.408	0.058	0.381	0.048	0.388	0.037	0.341	0.029	0.172	0.025	0.468	0.053	0.428	0.086
.dRepairGA	0.170	0.026	0.181	0.032	0.422	0.059	0.393	0.038	0.386	0.045	0.356	0.037	0.181	0.043	0.438	0.031	0.418	0.047
.dRepairRIGA	0.140	0.028	0.152	0.017	0.366	0.033	0.346	0.028	0.323	0.037	0.315	0.029	0.154	0.031	0.448	0.020	0.341	0.053
.dRepairHyperM	**0.059**	0.010	0.131	0.019	0.358	0.049	0.341	0.039	0.326	0.047	0.286	0.035	**0.067**	0.014	0.413	0.032	0.257	0.042
.dRepairGA-OOR	0.164	0.031	0.177	0.034	0.395	0.048	0.391	0.045	0.386	0.037	0.352	0.035	0.179	0.047	0.422	0.037	0.449	0.075
.dRepairRIGA-OOR	0.143	0.024	0.154	0.028	0.361	0.051	0.352	0.035	0.350	0.032	0.302	0.022	0.153	0.034	0.449	0.017	0.339	0.051
.dRepairHyperM-OOR	0.062	0.011	0.131	0.019	0.339	0.038	0.342	0.040	0.330	0.034	0.281	0.036	**0.068**	0.015	0.397	0.038	**0.242**	0.038
.Genocop	0.177	0.031	**0.059**	0.039	0.041	0.009	0.407	0.073	0.296	0.050	0.281	0.050	0.230	0.052	0.408	0.043	0.446	0.095
.GenocopwUPGwNRR	0.701	0.100	0.378	0.121	0.293	0.105	0.809	0.175	0.557	0.076	0.725	0.397	0.257	0.092	0.682	0.141	1.273	0.160
.GenocopwUPGwRR	0.638	0.178	0.440	0.279	0.294	0.116	0.688	0.187	0.593	0.176	0.578	0.189	0.440	0.123	0.784	0.093	1.356	0.193
.GenocopwUPCwNRR	0.138	0.030	0.074	0.025	**0.036**	0.008	**0.319**	0.074	0.280	0.049	0.291	0.061	0.171	0.033	0.427	0.039	0.447	0.101
.dGenocop	0.140	0.043	0.114	0.025	0.315	0.063	0.334	0.085	**0.263**	0.042	**0.242**	0.041	0.192	0.054	0.415	0.039	0.416	0.085

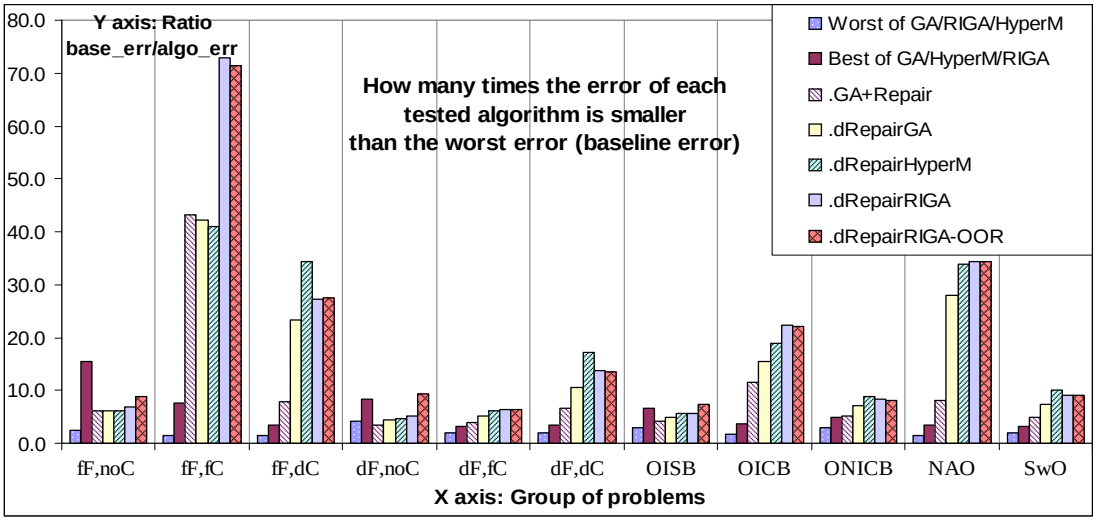

Figure 6.2: This figures shows the performance of dRepairGA-based variants compared with existing dynamic optimisation algorithms (the worst and the best of GA/RIGA/HyperM) and existing CH method combined with basic GA (Ga+Repair) in different group of problems. To avoid making the graph too cluttered we do not include dRepairGA_OOR and dRepairHyperM_OOR in the figure because their behaviours/performance are roughly the same as dRepairRIGA_OOR. We also do not include all versions of GA/RIGA/HyperM but only their worst and best performance. Instructions of how to read this figure can be found in the caption of Figure 5.3..

6.2.4 dRepairGA vs existing algorithms on unconstrained problems (dynamic and static)

dRepairGA-based algorithms vs existing DO algorithms

By comparing the bars of dRepairGA-based algorithms with the bars of existing DO algorithms in the group of *unconstrained static* problems (fF+noC) in Figure 6.2 (page 182), we can see that the dRepairGA algorithm perform better than the worst version of GA/RIGA/HyperM, but worse than the best version of GA/RIGA/HyperM by factors of 2.14 (dRepairGA vs RIGA-Elit) to 1.55 (dRepairHyperM vs RIGA-Elit). This is due to that in each generation dRepairGAs might need twice (or more) the number of evaluations than GA/RIGA/HyperM. This shows the trade-off that we need to pay if we want to have better performance in DCOPs: the algorithm might perform worse in unconstrained static problems.

In the group of *unconstrained dynamic* problems (dF+noC), the bar-comparison in Figure 6.2 also shows that dRepairGA algorithms perform worse than the best version of GA/RIGA/HyperM. However, I found that the ineffectiveness of algorithms using repair method in these problems,

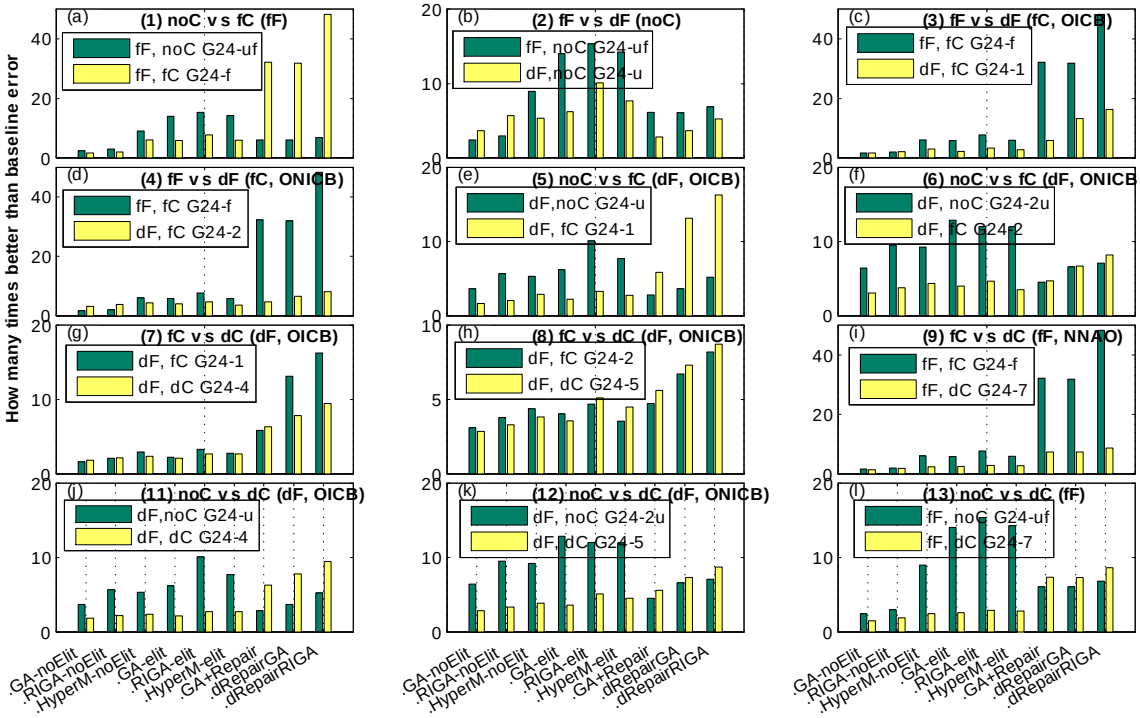

Figure 6.3: This figure summarises the effect of twelve different problem characteristics on the performance of existing DO algorithms (GA, RIGA, HyperM), existing CH algorithm (GA+Repair), and the newly proposed dRepairGA and dRepairRIGA (we do not include dRepairHyperM in the figure because its behaviour/performance is roughly the same as dRepairRIGA). Instruction to read this figure can be found in the caption of Figure 5.4 (page 122).

which have optima in boundaries of the search region, is due to the shortcomings of the repair operator in finding optima in boundaries of the search region (as already explained in Subsection 6.1.3). When we resolved the drawback of repair method by allowing the algorithm to search out-of-range (dRepairGA_OOR, dRepairRIGA_OOR, dRepairHyperM_OOR), the bar-comparison in Figure 6.2 shows that the out-of-range versions of dRepairGA perform better than GA/RIGA/HyperM by a factor of 1.22 to 2.22. This shows that the proposed algorithms are still useful when they are used to solve unconstrained dynamic problems. It is interesting to know that although they may need twice (or more) the number of evaluations than GA/RIGA/HyperM per generation, they are still able to perform better or equally in unconstrained dynamic problems.

dRepairGA-based algorithms vs existing CH algorithms

The bar-comparison in the group of *unconstrained static* problems (fF+noC) in Figure 6.2 shows that dRepairGA performs exactly the same as GA+Repair (their bars have almost the same heights). The result proves that the newly proposed dynamic CH mechanisms do not affect the

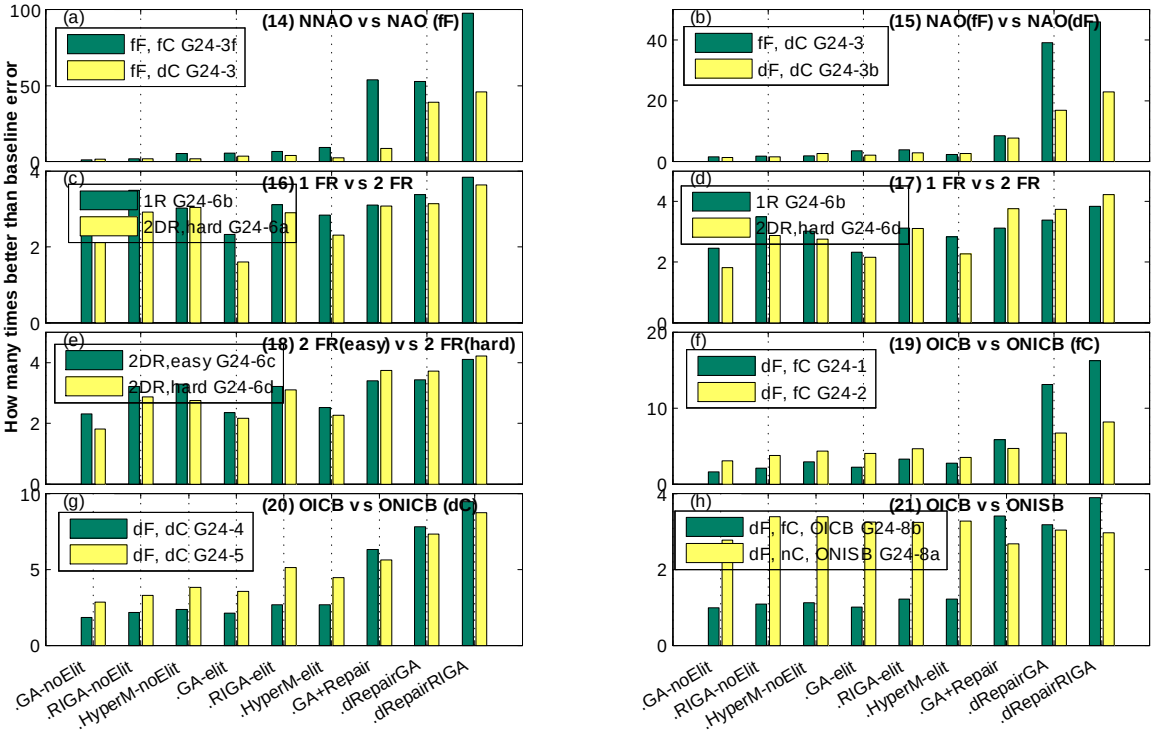

Figure 6.4: This figure summarises the effect of the other eight different problem properties on the performance of GA, RIGA, HyperM, GA+Repair, dRepairGA and dRepairRIGA. Instruction to read this figure can be found in the caption of Figure 5.4 (page 122).

repair method in solving static unconstrained problems. The figure also shows that the bars of dRepairRIGA/dRepairHyperM are slightly higher than that of dRepairGA, meaning that the additions of the RIGA and HyperM mutation strategies to the new algorithm offer a slight improvement on the performance of dRepairGA compared to the original GA+Repair. This shows that diversity has another usefulness for the repair method. When we apply out-of-range search, the OOR version of dRepairGAs offers a slight positive effect on the performance of dRepairRIGA (increase the performance by a factor of 1.29). This is due to some problems in the group fF+noC having a condition suitable for out-of-range search: their global optima in the boundaries of the search region.

The bar-comparison in the group of *unconstrained dynamic* problems (dF+noC) in Figure 6.2 shows that all versions of dRepairGA perform better than GA+Repair by an average factor of at least 1.33. The results suggest that the proposed mechanisms are effective in helping the algorithm to deal with the dynamics. The addition of RIGA and HyperM mutations for more diversity proves to be even more useful (better than GA+Repair by an average factor of 1.57 and 1.48, respectively). The addition of the OOR mechanism also helps improve the performance

because some problems in this group also have global optima in the boundaries of the search region.

Our experimental results (Table 6.2 and Figure 6.10, page 204 - to be introduced later) also shows that when being compared against Genocop III, our modified version dGenocop also behaves the same as dRepairGA when dRepairGA is compared against GA+Repair. Specifically, dGenocop also has the same performance as Genocop III in solving *unconstrained static* problems (fF+noC) and better performance than Genocop III in solving *unconstrained dynamic* problems (dF+noC). This result proves that the proposed mechanisms can be used effectively for solving unconstrained problems in not only dRepairGA but also other algorithms.

6.2.5 dRepairGA-based algorithms vs existing algorithms on static problems with constraints

When being compared with existing DO algorithms, the bar-comparison in Figure 6.2 shows that the CH technique (repair operator) used in dRepairGA-based algorithms helps them to work really well in the group of *static constrained* problems (fF+fC) and significantly outperforms all the tested existing DO algorithms. Specifically, dRepairGA-based algorithms perform better than basic GA by an average factor of 32.3 - 56.6, while the best results that GA/RIGA/HyperM can get for this group of problems is only better than basic GA by a factor of 5.16 (RIGA-Elit).

When being compared with existing CH algorithm (GA+Repair), the bar-comparison in Figure 6.2 shows that the introduction of the change-detection mechanism and update mechanisms in dRepairGA algorithms does not affect the efficiency of the repair method in solving *static constrained* problems (fF+fC). dRepairGA and GA+Repair have almost identical performance in this group of problems, with GA+Repair having an insignificantly better score (by a factor of 1.02). The very slightly worse performance of dRepairGA compared to GA+Repair might be due to the fact that dRepairGA might need to re-evaluate its best individual at every generation.

Similarly, the introduction of the change-detection mechanism and update mechanisms in dGenocop also has little impact on the efficiency of the algorithm in solving static constrained problems. The bar-comparison in Figure 6.10 (page 204) shows that dGenocop has almost the same performance as the original Genocop III in solving static constrained problems (Genocop III is slightly better by a factor of 1.06).

The introductions of a diversity-maintaining mechanisms (RIGA) into dRepairGA, again

make the algorithm work better by a factor of 1.75. This means that high diversity is useful not only for DO but also for CH techniques like the repair method. I will analyse this behaviour later in Section 6.3.2. The introduction of HyperM, as expected, does not offer any improvement because the hyper-mutation is not triggered in problems with static objective function and static constraints.

6.2.6 dRepairGA-based algorithms vs existing algorithms in DCOPs

As the new dRepairGA-based algorithms are designed to solve DCOPs, the result that is of most interest is the comparison between the new dRepairGA-based algorithms and existing algorithms in solving the class of DCOPs (dF+fC, fF+dC, dF+dC, OICB, ONICB, NAO and SwO).

dRepairGA-based algorithms vs existing DO algorithms

The summary result (bar-comparison) in Figure 6.2 (page 182) shows that dRepairGA works significantly better than all existing DO algorithms that I have tested in this chapter (by factors from 1.75 to 56.6).

The group of problems where dRepairGA-based algorithms work best are problems with fixed objective function and dynamic constraints (fF dC), especially those with moving constraints which expose new, better optima (NAO). This might be due to the usefulness of the repair method in tracking the moving feasible regions (I will have a more detailed analysis about this in Section 6.3).

dRepairGA-based algorithms also work very well in problems with dynamic objective functions and dynamic constraints (dF dC), again possibly due to its advantages in tracking the moving feasible regions, compared to other GA-based algorithms

The performance gap between dRepairGA-based algorithms and existing GA-based DO algorithms become smaller in the group of problems with fixed constraints and dynamic functions (dF fC). However, here the difference is still significant: dRepairGA algorithms perform better by factors from 1.75 to 3.11.

When we look at how the algorithms perform on problems with different characteristics, Figure 6.2 shows that the dRepairGA algorithms perform better than existing GA-based algorithms in all of the tested DCOP characteristics, of which the gaps are particularly large in problems with newly appearing optima (NAO), problems with optima in constraint boundaries (OICB) and problems with switching optima (SwO). This suggest that as expected, the combination of

repair method and our adaptive change-detection/update mechanism are useful at tracking the moving feasible regions, finding optima in boundaries and distributing individuals effectively when the optima switch between disconnected feasible regions.

There is one type of problem characteristic where the gap between dRepairGA algorithms and existing GA-based algorithm is less significant (although the former are still better by factors from 1.5 to 3.1). This is the group of problems where the optima are not in constraint boundaries (ONICB). This decrease in efficiency is caused not by the newly proposed mechanisms, but by the nature of the repair method because GA+Repair also show the same decrease in performance. This result suggests that the repair method might become less effective in solving problems with optima not in constraint boundaries. Nevertheless, the result shows that in spite of this decrease in performance, dRepairGA-based algorithms still perform significantly better than existing DO algorithms in the ONICB group.

It should be noted that there is another type of problem where the performance of dRepairGA-based algorithms are roughly equal to existing algorithms. This is the group of problems with optima in search boundaries (OISB). However, all problems in this group are not DCOPs but either static unconstrained or static constrained problems.

dRepairGA vs existing CH algorithms

The bar-comparison in Figure 6.2 shows that dRepairGA-based algorithms also perform significantly better than GA+Repair in all DCOPs (by factors from 1.34 to 4.53). The results prove that the proposed mechanisms work effectively in improving the drawbacks of GA+Repair in solving DCOPs.

The group of problems where the difference between dRepairGA-based algorithms and GA+Repair becomes largest are again problems with fixed objective function and dynamic constraints (fF dC) (by factors of 2.67 - 4.53) especially those with moving constraints which expose new, better optima (NAO). The difference between dRepairGA and GA+Repair in other problem groups are smaller, but still significant(by factors of 1.34 - 2.96). A more detailed analysis about the impact of each proposed mechanism on these improvements will be carried out in Section 6.3).

The results also show that basically dRepairGA-based algorithms also have the same behaviour as GA+Repair in DCOPs, only that they are able to achieve better performance. This

observation suggests that the efficiency of the original repair operator of GA+Repair in handling constraints is not affected by the newly proposed mechanisms.

I also observed that the improvement in performance of dGenocop over Genocop is very similar to the improvement of dRepairGA over GA+Repair. Specifically, dGenocop also has better performance than Genocop III in all groups of DCOPs and the behaviour of dGenocop is also the same as that of Genocop III. A detailed analysis about the advantages of dGenocop over Genocop III will be provided in Section 6.3.3.

6.2.7 Other interesting characteristics of dRepairGA-based algorithms

In this subsection I will take a further analysis of the behaviours of dRepairGA-based algorithms by looking at how different problem characteristics would affect their performance. This analysis was done based on observations on detailed results in Figure 6.3 and Figure 6.4 where the algorithms were tested in pairs of problems, of which the two problems of each pair are almost identical except that one has a particular characteristic and one does not.

The experimental results in these pairs of problems reveal some interesting behaviours of dRepairGA-based algorithms as follows.

First, although the presence of constraints makes the problems more difficult for existing DO algorithms, things are different for algorithms using repair methods. For these algorithms, the presence of constraints actually makes the problems (both static and dynamic) easier to solve. Evidence for this can be seen in all pairs that have two almost identical problems, one with constraints and the other without constraints. They are pair 1 (plot a), pair 5 (plot e), pair 6 (plot f), pair 11 (plot j), pair 12 (plot k) and pair 13 (plot l) of Figure 6.3 and pair 21 (plot h) of Figure 6.4. In all these plots we can see a distinct difference between existing GA-based DO algorithms and repair-based algorithms like GA+Repair and dRepairGA variants. This is the fact that while the presence of constraints make existing GA-based DO algorithms less effective compared to the unconstrained cases (in each of the aforementioned subplots their "fC" or "dC" bars are always lower than their "noC" bars), it also make repair-based algorithms become more effective in the constrained cases (in each of the aforementioned subplots the "fC" or "dC" bars of repair-based algorithms are always higher than their "noC" bars). The reason is that the presence of constraints helps increase selection pressures in the *repair routine.* This routine selects newly repaired solutions only if they are feasible. Otherwise, the repair process

is repeated until a feasible solution is produced (see step 2 of the Repair routine (Algorithm 9, page 151)). Obviously this type of selection is only meaningful in case there are constraints. In the unconstrained case, any repaired solutions will be accepted and hence there is no pressure for selection in the repair process, leading to slower convergence speed.

Another interesting, counter-intuitive observation is found in problems where the global optimum switches between disconnected feasible regions. Problems like these are supposedly more difficult to solve than problems with no disconnected feasible regions and the test results (to be explained in detail in Subsection 6.3.1) confirm that this assumption is true for existing DO algorithms. However, for dRepairGA-based algorithms, I observe that the presence of disconnected feasible regions does not always have any negative impacts on the performance. Instead, it might even help the algorithms to travel from one region to another faster. In addition, the larger the infeasible barrier between two feasible regions (supposedly harder to solve for algorithms using normal mutations), the better the performance of repair-based algorithm. Evidence can be seen in pair 17 (plot d) of figure 6.4 where repair-based algorithms have better performance in the disconnected-feasible-region case than in the single-feasible-region case. More evidence is given in pair 18 (plot e) of Figure 6.4 where the more isolated the disconnected feasible regions, the better the performance of repair-based algorithms. It should be noted that repair-based algorithms do not always perform better in problems with disconnected regions. One example is in pair 16 (plot c) of Figure 6.4 where the performance of repair-based algorithms in the disconnected-region case are only equal to or slightly worse than their performance in the single-region case. However, even in this pair of problem, the impact of disconnected-region on the performance of repair-based algorithms is much less than its impact on existing DO algorithms.

The reason for the effectiveness of repair-based methods in solving problems with disconnected-feasible regions will be analysed in Subsection 6.3.1.

6.3 What makes dRepair-based algorithms work well in DCOPs - a further analysis

In this section I will carry out a detailed analysis to investigate which factors have made the proposed algorithms work well in DCOPs (and why) and whether these factors are the results of our proposed mechanisms.

6.3.1 What makes dRepairGA-based algorithms better than GA/RIGA/HyperM in solving DCOPs

Ability to retain diversified solutions even if they are infeasible

The experimental results show that one of the reasons for dRepairGA-based algorithms to work better than GA/RIGA/HyperM (when these algorithms are combined with penalty functions) is that diversified but infeasible solutions are accepted with a higher percentage thanks to the way the repair method works. This helps dRepairGA-based algorithms maintain a higher level of diversity and hence might be able to react to changes in or near infeasible regions more effectively. This is one of the fundamental differences between the newly-proposed repair-based algorithms and the tested existing DO algorithms+penalty functions, which have a disadvantage of not being able to retain many diversified solutions if they are infeasible, as will be shown below.

Evidence for this advantage of dRepairGA-based algorithms can be seen in Table 6.3 (page 191) where I analysed the relationship between the percentage of infeasible areas over the total search area and the actual percentage of infeasible solutions over the total number of solutions selected for the next generation in the tested algorithms. To undertake this analysis I used a measure, the *percentage of selected infeasible individuals* proposed in Subsection 5.3.2. Among the individuals selected for the next generation, this measure counts the percentage of those that are infeasible. The average score of this measure (over all tested generations) is then compared with the percentage of infeasible areas over the total search area of the landscape. If the considered algorithm is able to treat *infeasible* diversified individuals and *feasible* diversified individuals on an equal basis (and hence to maintain diversity effectively), the two percentage values should be equal.

As can be seen in Table 6.3, dRepairGA-based algorithms have much higher percentage of infeasible individuals (43.8%-50.8%) than their GA-based elitism counterparts (12.5%-26.3%) although they use the same elitism mechanism. This means that dRepairGA-based algorithms is able to maintain more diversified but infeasible individuals. It should be noted that although the non-elitism versions of GA/RIGA/HyperM also have a nearly as high percentage of infeasible individuals as dRepairGA-based algorithms, as already discussed in Subsection 5.3.3, in non-elitism GA/RIGA/HyperM, this high score comes with a trade-off of slower convergence which eventually decrease the performance of the algorithms. The results in Table 6.2 (page 181) also

Table 6.3: The average *percentage of selected infeasible individuals* over all problems for each tested algorithm. The last row shows the average *percentage of infeasible areas* over all problems.

Algorithms	Percent of infeasible solutions
.GA-elit	12.5%
.RIGA-elit	26.3%
.HyperM-elit	14.8%
.GA-noElit	41.8%
.RIGA-noElit	46.8%
.HyperM-noElit	42.8%
.dRepairGA	43.8%
.dRepairRIGA	44.9%
.dRepairHyperM	48.0%
.dRepairGA-OOR	46.0%
.dRepairRIGA-OOR	48.3%
.dRepairHyperM-OOR	50.8%
.GA+Repair	47.0%
.GENOCOP	40.8%
.dGENOCOP	41.5%
Percentage of infeasible areas	**60.8%**

show that non-elitism GA-based algorithms have much worse performance than their elitism counterparts.

It is also interesting to see that although not equipped with an enhanced diversity maintaining mechanism such as random-immigrant or hyper-mutation, dRepairGA and GA+Repair still achieve a percentage of infeasible individuals as high as dRepairRIGA and dRepairHyperM. This confirms that the advantage of keeping infeasible solutions is due to that the repair method accept both infeasible solutions and feasible solutions, provided that they can contribute to the search.

Ability to track the moving feasible regions

Another reason for dRepairGA-based algorithms to perform better than GA/RIGA/HyperM is that they are able to track the moving feasible regions. As been briefly mentioned in Subsection 5.5.2, in order to track the moving optima successfully, it might be necessary to track the moving feasible regions where the optima are in first because in DCOPs the global optimum always either move along with the feasible regions or appear in a new feasible region.

To compare the performance of dRepairGA-based algorithms against other algorithms in tracking moving feasible regions, in this subsection I will analyse the ability of the tested algorithms in tracking the *optimal region*, i.e. the feasible region where the global optimum is currently in, and then evaluate the correlation between algorithms' tracking ability and the

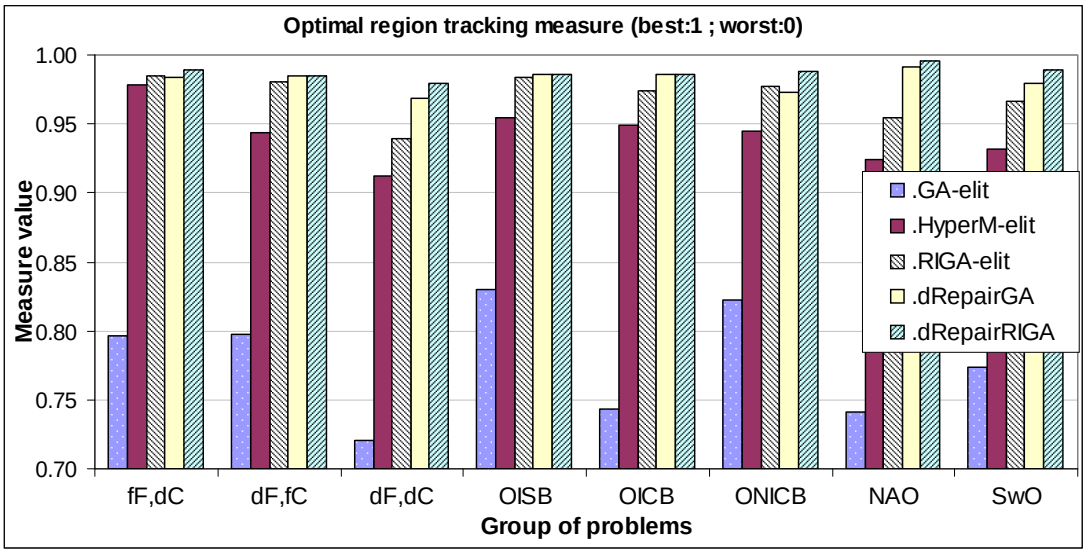

Figure 6.5: This figure shows how well each of the tested algorithms does in tracking the moving optimal feasible regions in different groups of DCOPs. The algorithms are evaluated using the score *optimal region tracking* (ORT). The ORT measure would be equal to 1 in the best case when the tested algorithm is able to track the region containing the global optimum at every generation, and would be equal to zero in case the algorithm is not able to track this region at all. This score is represented in the vertical axis. Explanations for the abbreviations in the name of problem groups can be found in Table 5.5 (page 108). It should be noted that in this figure we do not include the following groups of problems: (1) fF+fC because these problems are static; and (2) fF/dF+noC because there is no constraint.

speed and accuracy of their convergence. To evaluate the ability to track the optimal region, I propose a new measure: the *optimal region tracking measure* (ORT), which is calculated as the ratio between (a) the number of generations at which the algorithm has at least one individual in the optimal region and (b) the total number of generations:

$$ORT = \frac{\sum_{i=1}^{m} \sum_{j=1}^{p(i)} n\left(i,j\right)}{\sum_{i=1}^{m} p\left(i\right)} \tag{6.2}$$

where $n\left(i,j\right)$ is equal to 1 if the tested algorithm has at least one individual in the optimal region at the jth generation of the ith change and is equal to 0 otherwise; m is the number of changes and $p\left(i\right)$ is the number of generations at each change period i.

The ORT measure would be equal to 1 in the best case when the tested algorithm is able to track the region containing the global optimum at every generation, and would be equal to zero in case the algorithm is not able to track that region at all.

The summarised ORT scores of some algorithms are given in Figure 6.5 (page 192). To

avoid the graph to be too cluttered, I do not include the scores of the non-elitism versions of GA/HyperM/RIGA because they are much worse than the elitism versions. I also do not include the score of dRepairHyperM and the out-of-range (OOR) versions of dRepairGA-based because these scores are similar to the score of dRepairRIGA.

As can be seen in Figure 6.5, the ORT scores of existing DO algorithms are lower than that of dRepairGA-based algorithms, meaning that the latter are better in tracking the moving feasible regions. The difference becomes more significant in two cases: problems where the objective functions are dynamic (dF+fC, dF+dC and SwO) and problems where the moving feasible regions expose new, better optima without changing the value of the existing optimum (NAO). The reason for existing DO algorithms to not work well in the first case is that the global optimum does not gradually move but jumps from one region to another and hence it is difficult to track its movement using existing DO's tracking methods. The reason for existing DO algorithms to not work well in the second case is that the algorithms might not be aware of the newly appearing optima.

It is also interesting to note that even when being used without such diversity-maintaining mechanisms as random-immigrant or hyper-mutation, dRepairGA can still track the moving feasible regions very well. Figure 6.5 shows that the score of dRepairGA is almost the same as that of dRepairRIGA (only slightly lower in a few cases). It means that the repair operator and our three repair-based routines proposed in Subsection 6.1.3 are those that plays the major role in helping the algorithms to track the moving feasible regions. It also confirms our hypothesis in Subsection 6.1.2 that the original repair operator can be modified to track moving feasible regions.

Ability to travel between disconnected feasible regions

The third reason for the good performance of dRepairGA-based algorithms is that, different from the tested DO algorithms+penalty functions, dRepairGAs can travel easily through the infeasible areas separating disconnected feasible regions. This helps dRepairGAs to follow the global optimum when it switches from one region to another.

Evidence for this advantage of repair-based methods over existing DO algorithms can be found in the test cases 17, 18 (plots d, e) in Figure 6.4, page 184 that we have discussed previously in Subsection 6.2.7. As can be seen in the plots, while GA/RIGA/HyperM perform

Table 6.4: The *triggered-time count* scores and the *detected-change count* scores of algorithms using the HyperM mechanism in problem G24_3.

Algorithms	G24_3 (NAO fF+dC)			
	Trigger Count		Detected Change Count	
	Value	stdDev	Value	stdDev
.HyperM-noElit	164.20	11.29	1.82	0.83
.HyperM_elit	0.00	0.00	0.00	0.00
.dRepairHyperM	11.67	0.84	11.00	0.00
.dRepairHyperM-OOR	11.67	0.92	11.00	0.00

NAO - Newly Appearing Optimum

fF+dC - fixed objective Function, dynamic Constraints

worse (their bars become lower) in case there are more barriers separating the two disconnected feasible regions, repair-based algorithms still achieve the same performance or even have a better performance (their bars become higher) when there are barriers or when the barriers become larger. This shows that the presence of such barriers do not have a negative impact on repair-based algorithms or in other words repair-based algorithms can travel through the infeasible path between feasible regions thanks to the way they accept infeasible solutions during their search process.

Ability to detect changes occurring in the infeasible areas

Another possible reason for dRepairGA-based algorithms to perform better than change-detection DO algorithms, which are originally designed to detect changes in feasible areas only, is that they can detect changes in infeasible areas as well. Existing change-detection DO algorithms like HyperM might not be able to detect changes in problems with moving feasible regions which expose newly, better optima as G24_3 because HyperM only focuses on detecting changes happening to the current global optimum in the feasible regions. In problems like G24_3 where a new, better optimum appears due to changes in infeasible regions, the algorithm cannot detect the changes and consequently underperforms. Hypothetically the new algorithms should do better because they have detectors near the boundaries of infeasible regions to detect the situations where the constrained region expands/shrinks/moves/appears/disappears.

To analyse if the newly proposed change-detection mechanism can help the algorithm to work better than HyperM in this situation, we compared HyperM with its repair-based variant: our newly proposed dRepairHyperM. In term of detecting changes, dRepairHyperM (and other dRepair-based algorithms) is similar to HyperM in that it uses the same method to monitor fitness drop to detect changes. However, dRepairHyperM also use our newly proposed mecha-

nism to monitor the shrink and expansion of infeasible regions to detect changes in infeasible areas. Details of this mechanism has been described in Subsection 6.1.3 and in the routine *DetectChange* in Algorithm 10 (page 167). This *DetectChange* routine is also used in other dRepair-based algorithms tested in this chapter.

To evaluate the ability of HyperM and dRepairHyperM in detecting changes, I used the two measures proposed in Subsection 5.3.2: the measure *triggered-time count*, which counts the number of times the hyper-mutation-rate is triggered by the algorithm, and the measure *detected-change count*, which counts the number of triggers actually associated with a change. For HyperM/dRepairHyperM, triggers associated with a change are those that are invoked by the algorithm within ν generations after a change, with ν is the maximum number of generations (five in our implementation) needed for HyperM/dRepairHyperM to detect a drop in performance. These two measures indicate how many times an algorithm triggers its hyper-mutation; whether each trigger time corresponds to a new change; and if there is any change goes undetected during the search process.

Evidence for the advantage of dRepairGA-based algorithms as dRepairHyperM over HyperM can be seen in Table 6.4 (page 194). The comparison results show that on the one hand, both the elitism and non-elitism versions of HyperM are not able to detect changes in G24_3 (the algorithm either is not able to trigger its hyper-mutation rate to deal with changes (elitism case, *triggered-time count*=0 and *detected-change count*=0) or is not able to trigger its hyper-mutation rate correctly when a change happens (non-elitism case, *triggered-time count*$\simeq$164 and *detected-change count*$\simeq$1.8)). On the other hand, dRepairHyperM and dRepairHyperM-OOR are able to detect all changes occurring during the search process (*triggered-time count*$\simeq$11 and *detected-change count*=11). This shows the advantage of dRepairGA-based algorithms in detecting changes.

6.3.2 What makes dRepairGAs/dGenocop better than GA+Repair/Genocop in solving DCOPs?

Ability to update reference individuals when changes happen

In algorithms using the original repair method such as GA+Repair and Genocop III, after a change the reference individuals might become out-of-date or even infeasible. These outdated individuals might mislead the algorithms and consequently affect the search performance. Our

hypothesis is that dRepairGAs/dGenocop might be better than GA+Repair/Genocop in this aspect because they are able to repair their reference individuals whenever a change happens thanks to the routine *UpdateReferencePop*. This advantage helps such algorithms as dRepairGAs/dGenocop to update their knowledge about the problem to react to the change better.

To analyse if the newly proposed change-detection mechanism can really help the algorithm to work better than existing CH algorithms in this situation, we compared GA+Repair with dRepairGA, where the original GA+Repair is combined with the newly proposed routines described in Subsection 6.1.3: *DetectChange* and *UpdateReferencePop*. These two routines are designed to adaptively update the reference population whenever a change happens.

To test if the algorithms are able to update the reference individuals properly, I used a measure proposed in Subsection 5.3.2: the *plot of number of reference individuals that are feasible*. If an algorithm is able to update the reference individuals properly, it should be able to maintain a reference population of all feasible individuals all the time during the search process and the plot diagram would show a flat, constant line.

The most suitable environments to test this behaviour of the two algorithms are DCOPs with dynamic constraints where after each change the previous best feasible solutions are hidden by the moving infeasible region. In the G24 benchmark set, the problems that have this property are the G24_4, G24_5 and G24_7. G24_4 and G24_5 belong to the problem group *dF,dC* while G24_7 belongs to the problem group *fF,dC* (see Figure 6.2). As can be seen in Figure 6.2, in both groups the performance of GA+Repair decreases significantly compared to the case where the constraints are fixed (fF,fC). In this analysis we will see if the moving infeasible region makes any of the reference individuals become infeasible. If no reference individual becomes infeasible, after each change the total number of feasible reference individuals should remain to be five (the size of the reference population). If one or more individuals do become infeasible, there should be a drop in the total number of feasible reference individuals.

The comparison results using the aforementioned measure are shown in Figure 6.6, page 197. The figure shows that, in all cases the original repair method (GA+Repair) is not able to keep all reference individuals feasible during the search. When a change happens, the number of feasible reference individuals drops to a very low level. The figure also shows that this drawback has been overcome in dRepairGA. We can see in the dRepairGA plot that when GA+Repair is combined with the *UpdateReferencePop* routine, all reference individuals are updated and

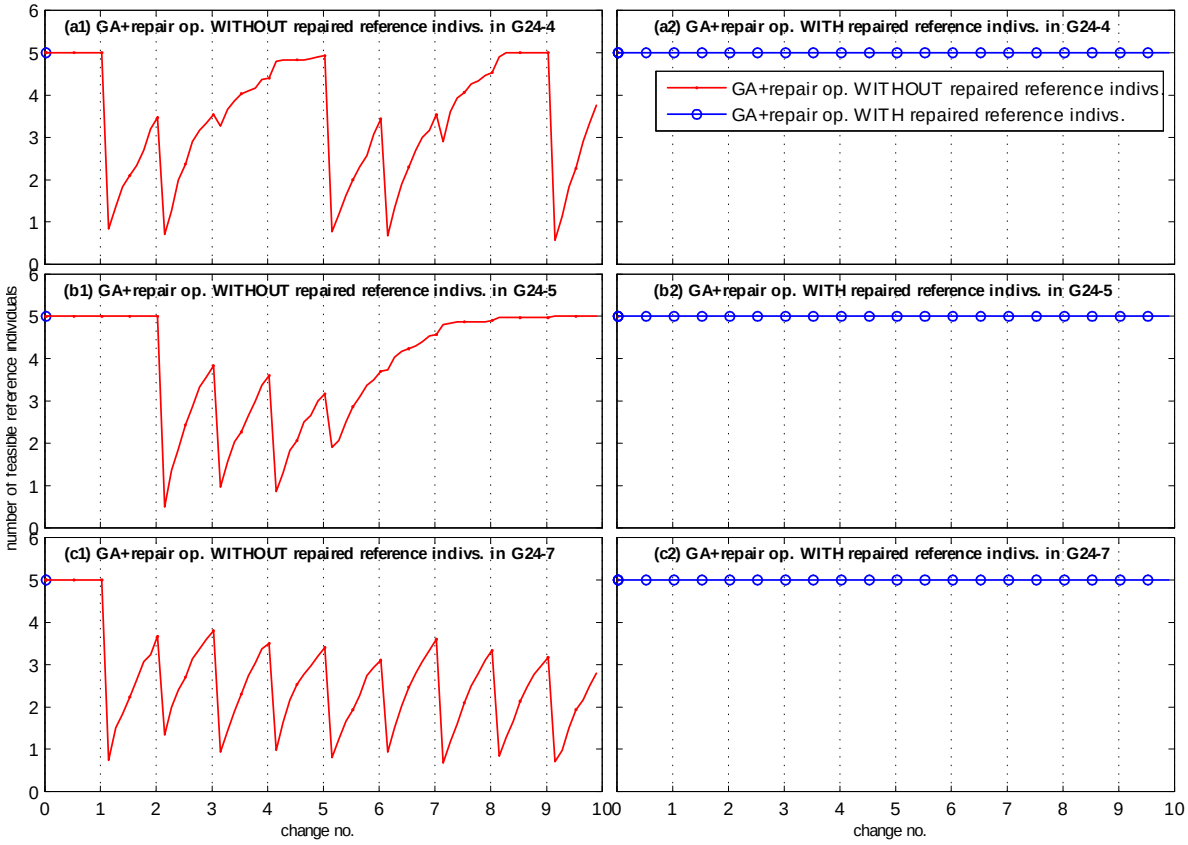

Figure 6.6: This figure shows a comparison of GA+Repair and dRepairGA in maintaining feasible reference individuals in problems with moving infeasible regions. The plot values (y-axis) show the number of reference individuals that are feasible. If no reference individual becomes infeasible, the plot should shows that after each change the total number of feasible reference individuals still remains to be five. On the left hand side is GA+Repair. On the right hand side is dRepairGA, which is the combination of GA+Repair and a special adaptive mechanism (the *UpdateReferencePop* routine - introduced in Algorithm 11, page 172) to make sure that the outdated reference individuals are updated whenever a change happens.

remained feasible through out the search process.

Later in Subsection 6.3.3 I will analyse the level of performance improvement that the use of the routine *UpdateReferencePop* can bring to the original Genocop III and GA+Repair.

Ability to adaptively balance feasibility and infeasibility

Another reason that makes dRepairGAs/dGenocop better than GA+Repair/Genocop is that they are able to adaptively balance feasibility and infeasibility better when changes happen. As mentioned previously in Subsection 5.4.4, existing CH strategies like the repair method in GA+Repair/Genocop might suffer from the outdated problem and hence might not be able to balance feasibility/infeasibility well in DCOPs.

To analyse if the newly proposed change-detection mechanism can really help improve balancing feasibility/infeasibility in DCOPs, we compared GA+Repair with the new dRepairGA algorithm. As already described in Subsection 6.1.4, dRepairGA is a combination of GA+Repair with the newly proposed change-detection (*DetectChange)* and update routines: *UpdateSearchPop* and *UpdateReferencePop*. These routines are designed to adaptively balance feasibility and infeasibility whenever a change happens.

To test if the performance of GA+Repair versus dRepairGA in balancing feasibility/infeasibility in dynamic environments, I used a measure proposed in Subsection 5.3.2: the *plot of number of feasible individuals in each disconnected feasible region* to monitor the number of feasible individuals in each disconnected feasible region and the ratio of feasibility/infeasibility. If the balancing mechanism works well in the DCOP case, it should be able to manage a good distribution of individuals so that the better feasible regions should have more feasible individuals.

The most suitable environments to test this behaviour are DCOPs with two disconnected feasible regions where the global optimum keeps switching from one region to another after each change or after some consecutive changes. In the G24 benchmark set, the problems that have this property are the G24_1, G24_2, G24_3b, G24_4, G24_5, G24_6a, G24_6c, G24_6d, and G24_8b where the global optimum switches from one region to another after each period of one or two changes. All these problems belong to the group *SwO* in Figure 6.2 (page 182), where we can see that the performance of GA+Repair significantly decreases compared to the stationary constrained case (fF, fC). In such SwO problems like these, if the balancing mechanisms of the tested algorithms work well, at each period between changes the algorithm should be able to focus most feasible individuals on the region where the global optimum is currently in while still maintaining the same ratio of feasibility/infeasibility for diversity purpose.

The *plots of number of feasible individuals in each disconnected feasible region* of GA+Repair and dRepairGA in these functions are given in Figure 6.7 (page 200) and Figure 6.8 (page 201). It should be noted that the measure scores of the tested algorithms in three problems: G24_2, and G24_6c/d are not shown in because they are similar to that of G24_5, and G24_6a, respectively.

The figures show that in all cases except G24_3b, the existing CH method GA+Repair is not able to focus most feasible individuals on the region where the global optimum is currently in. Instead, the majority of feasible individuals still remained in one single region (region 2), which

is where the global optimum firstly was before the changes happen. The number of individuals in the other region (region 1) remains low regardless of whether the global optimum has switched into the region or not. These results show that, due to its outdated information and strategy, the GA+Repair algorithm is not able to follow the switching optimum well.

Figures 6.7 and 6.8 also show that the drawback above has been resolved in the newly proposed dRepairGA. As can be seen in the figures, the presence of the routines proposed in Subsection 6.1.3 helps dRepairGA focus most of its feasible individuals to the region where the global optimum has moved into whenever a change happens while still maintaining the same ratio of feasibility/infeasibility for diversity purpose in all problems.

Later in Subsection 6.3.3 I will analyse the level of performance improvement that the routines *UpdateSearchPop* and *DetectChange* bring to the original Genocop III and GA+Repair.

More diversity helps the repair operator approach toward the global optimum faster

The third reason that might make the newly proposed algorithms as dRepairRIGA/ dRepairHyperM perform better than existing repair-based methods is the high-level of diversity. The summarised results in Figure 6.2 (page 182) shows that when being hybridised with diversity-enhanced mutation strategies such as RIGA and HyperM, the performance of dRepairGA can be significantly increased in all groups of problems.

This improvement is due to two factors. The first, and obvious factor is the benefit of diversity in dealing with environmental dynamics. Evidence of this advantage can be seen in the experiments in this chapter, for example in Table 6.2 where we can see that GA-based algorithms with high diversity as RIGA and HyperM perform better than basic GA in most dynamic problems.

I found that there is another interesting factor that helps dRepairRIGA/dRepairHyperM outperform dRepairGA. This is the fact that, besides its usefulness in handling dynamics, diversity also improves the effectiveness of the repair method in handling constraints. Evidence for this advantage can be found in Figure 6.2, the group of static constrained problems (fF,fC). In this type of problem, the performance of dRepairRIGA is significantly better than that of GA+Repair and dRepairGA. As the only difference between dRepairRIGA and GA+Repair/dRepairGA when solving static constrained problems is the level of diversity, the good performance of dRepairRIGA must be brought by its high level of diversity. In addition, because there is no

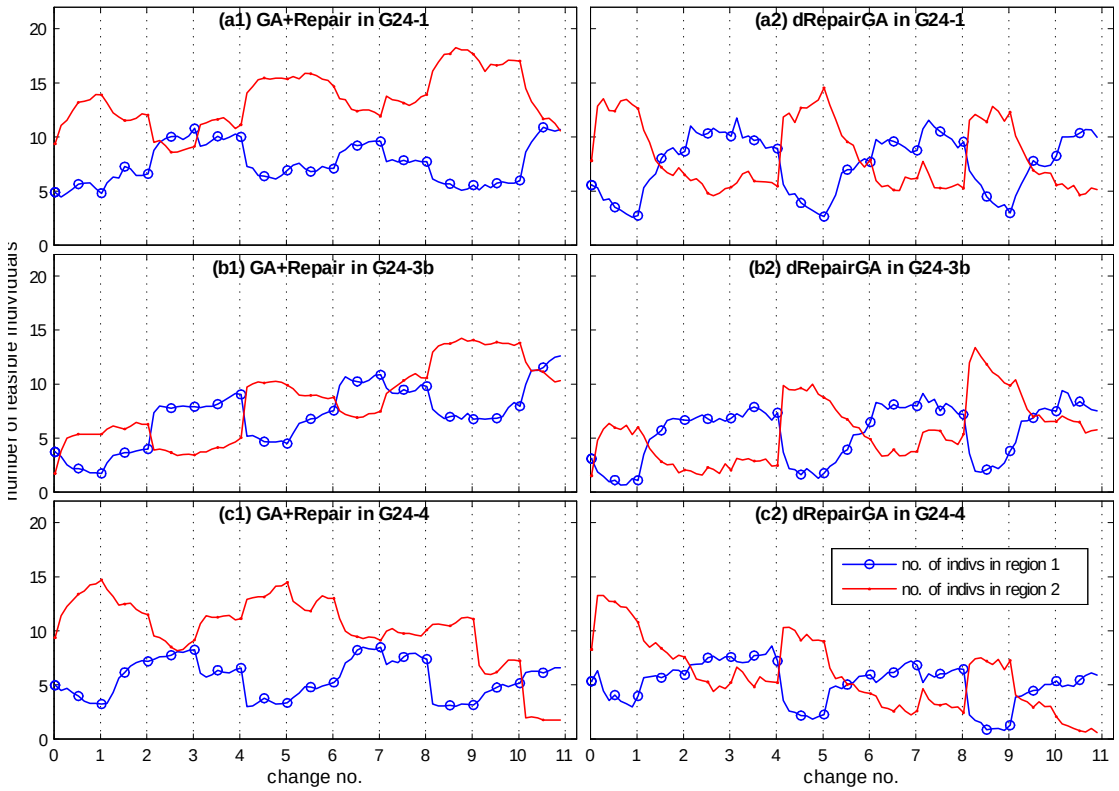

Figure 6.7: This figure shows how the balance strategies of GA+Repair (left) and dRepairGA (right) distribute their feasible individuals in the disconnected feasible regions of problems G24_1, G24_3b and G24_4. The problems tested in this figure are those with global optima switching between two disconnected feasible regions after each change or after a few changes. e.g. in G24_1 and G24_3b the location of the global optimum is as follows: change 1: region 2, change 2-4: region 1, change 5: region 2 and so on. The plot lines with circles show the number of feasible individuals in region 1, and the plain plot lines show the number of feasible individuals in region 2. If the balance strategy works well, most individuals should be focused on the region where the global optimum is currently in. It means that when the optimum switches to region 2, the number of individuals in region 2 should be high and the number of individuals in region 1 should be low. When the optimum switches back to region 1, the reverse thing should happen, i.e. number of individuals in region 1 should be high and that number in region 2 should be low.

environmental dynamic in static constrained problems, we can conclude that the benefit of high diversity in this type of problem is not to handle dynamics but to help the repair method to be more effective in handling constraints. The reason for the usefulness of diversity to the repair process is possibly due to that higher diversity might allow the repair process to approach the global optimum from more directions, and hence accelerate the search process.

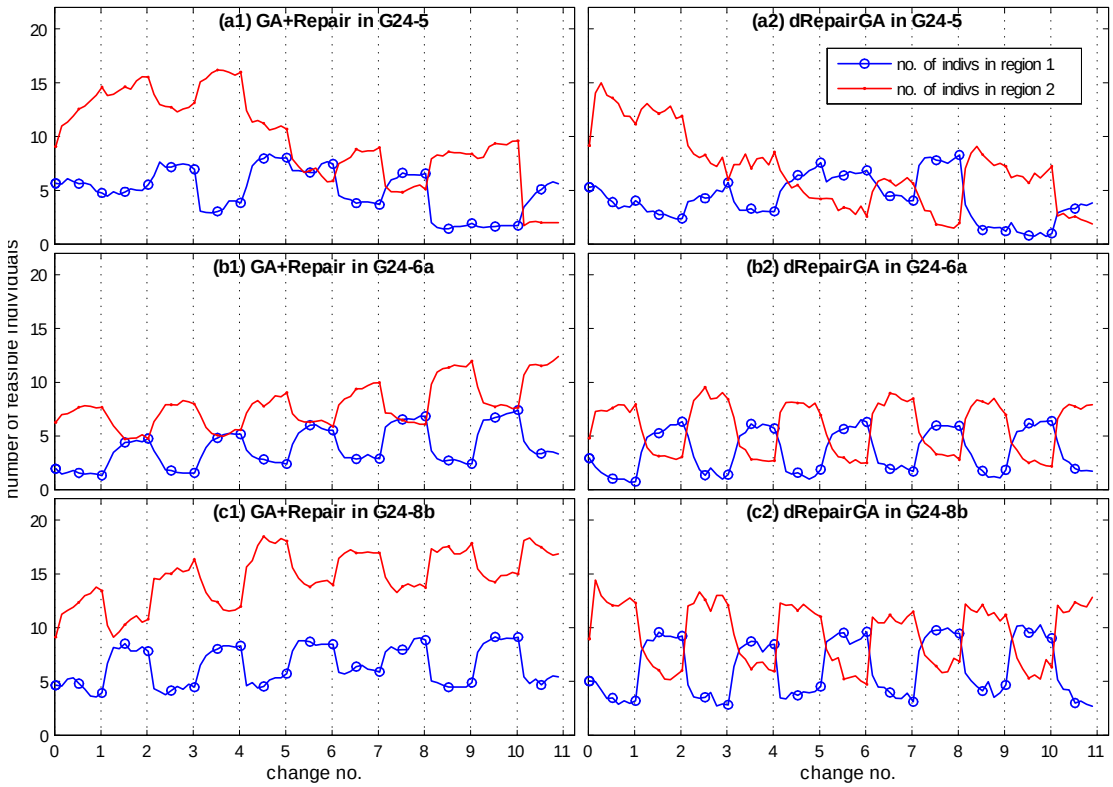

Figure 6.8: This figure shows how the balance strategies of GA+Repair (left) and dRepairGA (right) distribute their feasible individuals in the disconnected feasible regions of the other three problems: G24_6c, G24_6d and G24_8b. Instructions to read the figure can be found in Figure 6.7.

Ability to search out of range to find optima in search boundary faster

The ability to search out of range (OOR) of dRepairGA_OOR, dRepairRIGA_OOR, dRepairHyperM_OOR also brings them certain advantages to existing repair-based methods in solving problems where the global optima is in the boundaries of the search region. Experimental results in Figure 6.2 and in Table 6.2 show that compared to the original algorithms, the OOR versions achieve better performance in group of problems with optima in search boundaries (OISB, fF+noC, dF+noC) while still having almost equal performance in other group of problems. It means that the 25% out-of-range search mechanism is useful to find in-boundary optima and does not create any significant negative impact on the performance of algorithms in other cases.

6.3.3 The contribution of each component to the performance of dRepairGAs and dGenocop

Because the proposed dRepairGAs and dGenocop algorithms combine different components such as the four routines Repair, DetectChange, UpdateReferencePop, UpdateSearchPop; the RIGA/HyperM mutation strategies; and the out-of-range mechanism OOR, it is of interest to study the contribution of each component on the performance of the complete algorithms. Among these components, I have already analysed the impacts of the Repair routine by comparing the performance of GA+Repair with that of GA in Figure 6.2. I have also analysed the impact of the RIGA/HyperM mutation strategies and the OOR mechanism on algorithms' performance in Subsections 6.3.2 and 6.3.2, respectively.

In this subsection I will analyse the impacts of the rest of the proposed components - the three routines DetectChange, UpdateReferencePop and UpdateSearchPop - on the performance of GA+Repair and Genocop III. In order to carry out the analysis, I firstly integrated each of the above routines with the original version of GA+Repair and Genocop III to create a corresponding modified variation, then compared this modified variation with the original algorithm. The following combinations were tested:

- GA+Repair/Genocop + wNUwNRR (No Update, No Repair Reference population): These are the original versions of the constraint-handling algorithms. None of the newly proposed routines is integrated.
- GA+Repair/Genocop + wUPGwNRR (Update Per Generation, No Repair Reference population): No change-detection is made. Only the UpdateSearchPop routine is carried out at every generation
- GA+Repair/Genocop + wUPGwRR (Update Per Generation, Repair Reference population): No change-detection is made. Both the UpdateSearchPop and UpdateReferencePop routines are carried out at every generation
- GA+Repair/Genocop + wUPCwNRR (Update Per Change, No Repair Reference population): Change detection is enabled. The UpdateSearchPop routine is carried out adaptively whenever a change is detected.

- GA+Repair/Genocop + wUPCwRR (Update Per Change, Repair Reference population): Change detection is enabled. Both the UpdateSearchPop and UpdateRefererecePop routines are carried out adaptively whenever a change is detected.

Detailed comparisons on groups of problems for GA+Repair-based algorithms is given in Figure 6.9 and the same comparison for Genocop-based algorithms is given in Figure 6.10. It should be noted that because all GA+Repair variants are developed from GA+Repair, in Figure 6.9 I chose the original GA+Repair as the baseline to compare all GA+Repair variants. Likewise, because all Genocop-based algorithms are developed from Genocop III, in Figure 6.10 I chose the original Genocop III as the baseline to compare all Genocop-based algorithms.

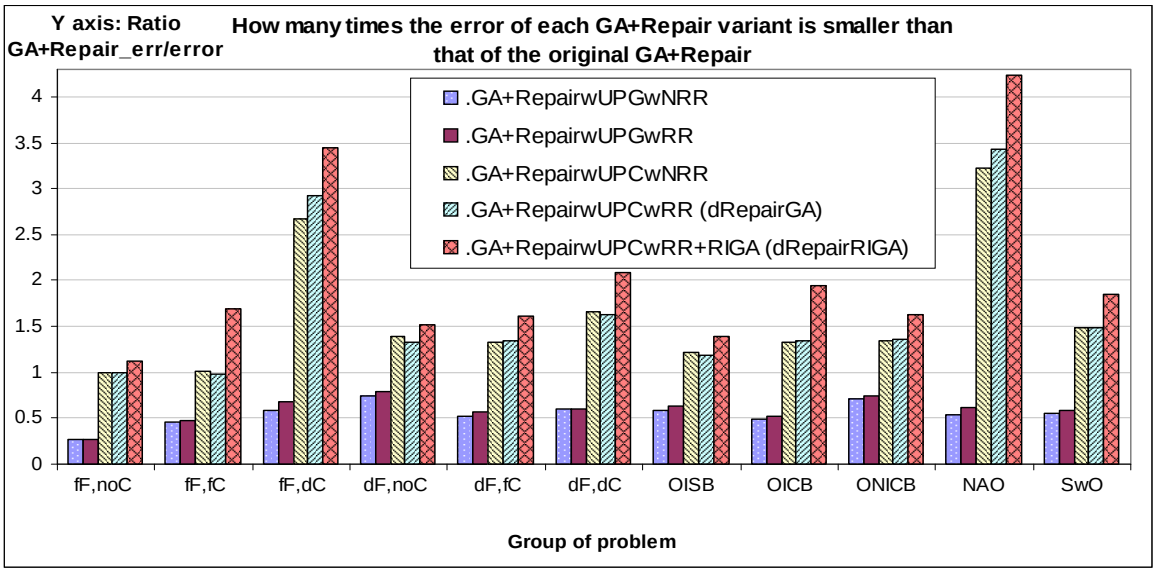

Figure 6.9: This figure shows the effect of each of our proposed adaptive balancing/updating mechanism on GA+Repair in solving DCOPs. Each algorithm in the graph represents a combination of each proposed mechanism and GA+Repair. The vertical axis show the ratio between the average (modified offline) error of the original GA+Repair and that of each algorithm. This ratio indicates how many times each algorithm performs better (ratio > 1) or worse (ratio < 1) than the the original GA+Repair in term of modified offline error (a ratio score of 1 means that there is no improvement nor decrease in performance). Explanations for the abbreviations in the name of problem groups can be found in Table 5.5 (page 108).

Generally, the comparison results in Figures 6.9 and 6.10 are as expected, i.e. each of the proposed components does have its own contribution to improve the performance of algorithms. Detailed analysis shows some interesting observations about the contribution of each component.

First, the results suggest that the process of updating algorithms' knowledge/strategy (as done in the UpdateSearchPop and UpdateRefererecePop routines) can only be effective if they

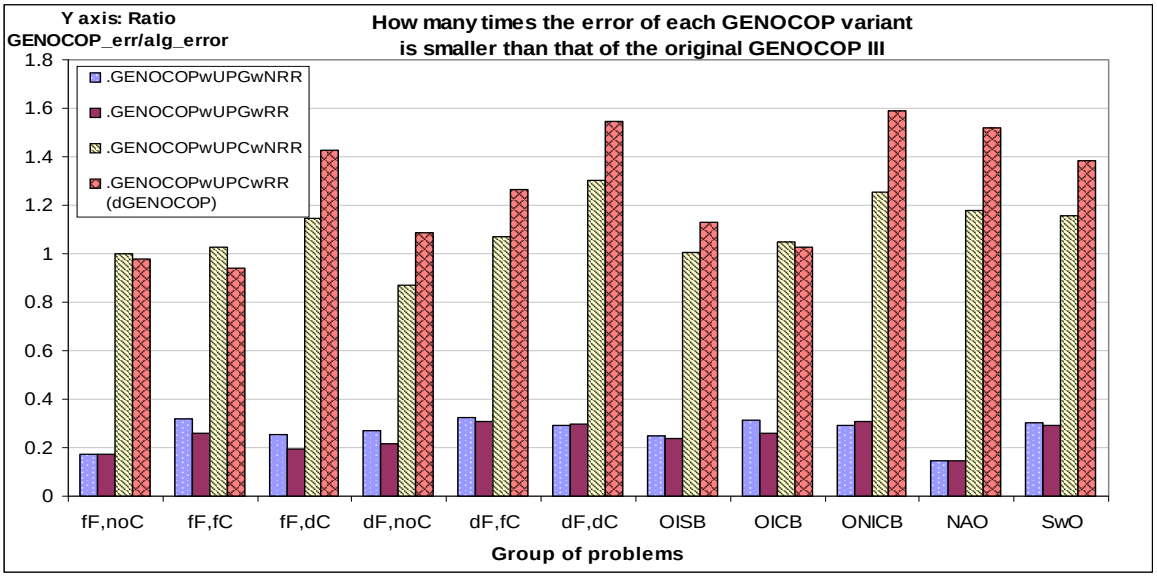

Figure 6.10: This figures shows the effect of each of our proposed adaptive balancing/updating mechanism on Genocop III in solving DCOPs. Each algorithm in the graph represents a combination of each proposed mechanism and Genocop III. The vertical axis show the ratio between the average (modified offline) error of the original Genocop III and that of each algorithm. This ratio indicates how many time each algorithm performs better (ratio > 1) or worse (ratio < 1) than the the original Genocop III in term of modified offline error (a ratio score of 1 means that there is no improvement nor decrease in performance). Explanations for the abbreviations in the name of problem groups can be found in Table 5.5 (page 108).

are combined with the adaptive change-detection method (routine *DetectChange*). Figures 6.9 and 6.10 shows that when the algorithms update their knowledge/strategy at every generation (GA+RepairwUPG and GENOCOPwUPG) instead of at the beginning of each appropriate change period (GA+RepairwUPC and GENOCOPwUPC), algorithm performance is decreased. This is due to that the update process requires additional function evaluations to be taken (in the routine *UpdateRefererecePop* (Algorithm 11, page 172) individuals are re-evaluated and in the routine UpdateSearchPop (Algorithm 12, page 172) individuals are repaired and hence also re-evaluated). If the update process is undertaken at every generation, it may take up too many evaluations and hence prevent the algorithms from doing the search effectively.

Second, the results show that when the UpdateSearchPop routine is combined with the adaptive change-detection method (as in GA+RepairwUPC and GENOCOPwUPC), the performance of the tested algorithms improves in all dynamic cases (fF+dC, dF+noC, dF+fC, dF+dC) and remains almost unchanged in the static cases (fF+fC, fF+noC). This proves that detecting changes and updating the search population are very useful in handling environmental

dynamics while not causing any negative impact on performance in the static cases.

Third, when comparing the performance of the -RR versions (Repair Reference population) of the algorithms with the -NRR versions (No Repair Reference population), the results in the two figures show that the process of updating reference individuals using the routine UpdateRefererecePop (denoted RR) is useful in dealing with problems with dynamic constraints (fF+dC, where the moving constraints might make existing reference individuals feasible) and problems with newly appearing optima (NAO, where existing reference individuals might not reflect the best region in the search space). These two types of problems are those where the reference individuals are most likely to become outdated after each change and hence as expected they are the types of problems where the performance has been improved most using the updateReferencePop routine. In other types of problems, there are improvements but slightly less significant (in Genocop-based algorithms) or there is no improvement (in GA+Repair-based algorithms). In a few cases the UpdateReferencePop routines may even decrease performance as in the problem group dF+noC (GA+Repair algorithms) and in the problem group fF+fC / OICB (Genocop algorithms) but the impact is insignificant compared to the improvements in other cases.

6.4 A detailed analysis on parameter values

As mentioned in Subsection 6.2.2, to maintain a fair comparison environment, in all previous experiments, if possible we set the parameters of all algorithms to the same values, which are the default or best reported values given in the original research of RIGA (Grefenstette 1992), HyperM (Cobb 1990) and Genocop III (Michalewicz n.d.).

However, it is unknown if these default or best reported parameter values are most suitable for solving DCOPs. To answer this question, in this section I will carry out a detailed analysis to investigate how different parameter values would affect the performance of all tested algorithms, including existing DO/CH algorithms and the newly proposed algorithms in solving the problems in the G24 benchmark set. The analysis will provide us with suggestions on how to choose the best parameter values to solve DCOPs. As the problems in the G24 benchmark set include not only DCOPs but also static unconstrained, dynamic unconstrained, and static constrained problems, this analysis will also show how robust the tested algorithms are using different parameter values.

It is also worth noting that for algorithms with both elitism and non-elitism versions as

GA/RIGA/HyperM, in this section I will only present the results of the elitism versions because in our previous experiments the elitism versions have better performance than the non-elitism versions.

6.4.1 Performance measures

In the previous experiments, it was possible to present the detailed performance of each algorithm in every problems of the benchmark set as well as in groups of problems because I used only one single set of parameter values. In the experiment in this section, however, it is impractical to present the results in such a detailed level because the amount of data is very large due to the fact that we are going to test the algorithms in all possible ranges of parameter values.

This limitation requires us to find a new type of measure which can accurately represent the overall performance of each algorithm in all tested problems under different parameter settings. To represent the overall performance of an algorithm in all tested problems, one solution is to calculate the average value of the errors from all problems (or calculate the ratio between the error and a fixed baseline error (*baseline-error-ratio*) as we did in our previous experiments), but the result might be biased toward larger errors. For example, if an algorithm has a large error of 1000 in problem A and a very small error of 1e-10 in problem B, the average value would be a large error of 500+5e-11. This large error obviously does not reflect the good performance that the algorithm has achieved in problem B. In cases where we only calculated the algorithm error in each single problem (like the results in Figure 6.3 (page 183) and Figure 6.4 (page 184)) or in groups of few problems with similar characteristics and similar problem structures (like the results in Figure 6.2 (page 182)), there is little to no bias, so we can use the baseline-error-ratio to evaluate algorithms performance. However, in cases where we want to evaluate the average error of all 18 problems in the benchmark set, the bias in errors might be significant and this bias will consequently make it very difficult to accurately evaluate the overall performance of the tested algorithms.

To overcome this limitation, in this section I will use a newly proposed measure, the *normalised score*, which evaluates the overall performance of an algorithm compared to other peer algorithms in solving a group of problems in a normalised way. The idea is that, given a group of n tested algorithms and m problems, for each problem j the performance of each algorithm is normalised to the range $(0, 1)$ so that the best algorithm in that problem j will have the score of

1 and the worst algorithm will get the score of 0. The final overall score of each algorithm will be calculated as the average of the normalised scores from each individual problem. According to this calculation, if an algorithm is able perform best in all tested problems, it would get an overall score of 1. Similarly, if an algorithm performs worst in all tested problems, it would get an overall score of 0.

Given a group of n tested algorithms and m problems, a formal description of the the *normalised score* of the ith algorithm is given in Equation 6.3:

$$S_{norm}(i) = \frac{1}{m}\sum_{j=1}^{m}\frac{|e_{\max}(j) - e(i,j)|}{|e_{\max}(j) - e_{\min}(j)|},\ \forall i = 1:n. \tag{6.3}$$

where $e(i,j)$ is the modified offline error of algorithm i in problem j; and $e_{\max}(j)$ and $e_{\min}(j)$ are the largest and smallest errors among all algorithms in solving problem j.

The advantage of the normalised score is that it is unbiased. The fact that an algorithm might get a very large or very small error on a particular problem (like the example given previously) would not bias the overall score as it does when we use the traditional mean value of errors.

In the experiments in this section, to evaluate the behaviour of algorithms from different perspectives I will present the performance of algorithms using both measures: the *baseline-error-ratio* used in previous experiments and the *normalised score*. The normalised score provides more accurate evaluation on the overall performance but the baseline-error-ratio would also be useful in highlighting the situations where an algorithm achieves a very good result with high precision (and hence high ratio score) in certain problems.

6.4.2 Crossover rate

The first parameter that we are going to analyse is the crossover rate. Figure 6.11 shows our analysis of how changing the crossover rate would affect the performance of the tested algorithms when all other parameters are kept constant. It should be noted that in this analysis I do not test the impact of changing crossover rate on Genocop III and dGenocop. The reason is that due to the special design in Genocop III/dGenocop changing the crossover rate would affect the rate of nine other specialised operators. In that case, any impact in performance might be caused by not only the change in crossover rate but also by the consequent changes in the probability rates of other specialised operators.

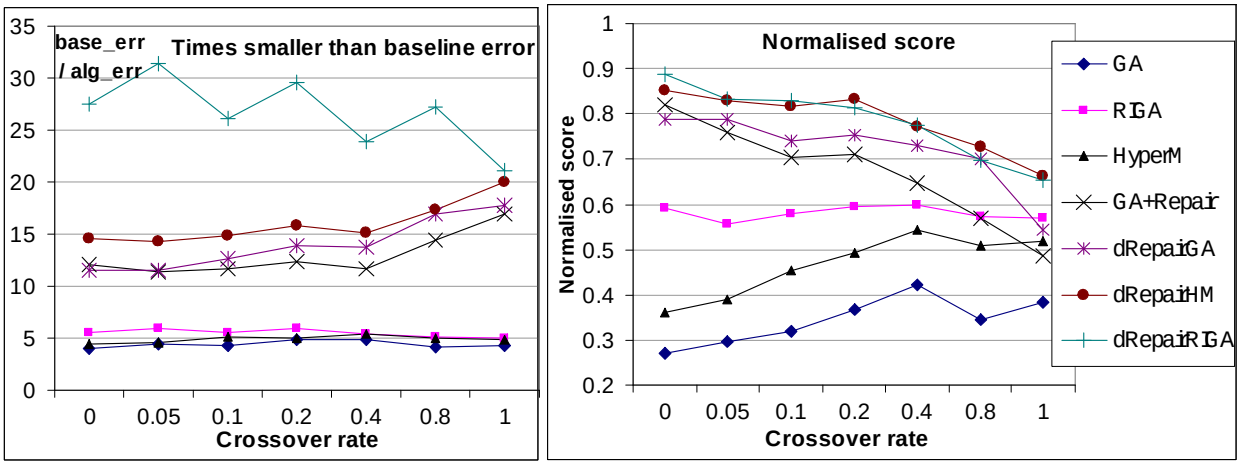

Figure 6.11: This figure shows an analysis of how changing the crossover rate would affect the overall performance of the tested algorithms on all the 18 tested problems. The impact of different crossover rates on algorithm performance is calculated using two performance measures. On the left is the average of the *baseline-error-ratio*, which represents how many times the error of an algorithm on a problem is smaller than the baseline error (the worst observed error). This measure can be misleading because its score would be bias toward a few problems where an algorithm has very small errors. However, the measure is somehow still useful because when we see an algorithm with a high *baseline-error-ratio*, we know that the algorithm must have had very small errors in at least one of the tested problems. On the right is the *normalised score* (see Equation 6.3), which provides more accurate evaluation on the overall performance and robustness of the tested algorithms. The higher the y-axis values, the better the performance.

Figure 6.11 shows both the average *baseline-error-ratio* scores (left) and the *normalised scores* (right) under different crossover rates. Some observations can be seen from the results in the figure. First, when being used to evaluate the overall performance in all problems, the *baseline-error-ratio* score can be misleading. Based on the *baseline-error-ratio* score (left plot) it looks as if (1) dRepairRIGA has the best overall performance and (2) the increase of crossover rate helps algorithms as dRepairHyperM, dRepairGA and GA+Repair increase their performance. However, the *normalised scores* (right plot) shows that these are not true. dRepairRIGA actually only has roughly the same overall performance as dRepairHyperM, and the increase of crossover rate actually has negative effect on the performance of dRepairRIGA, dRepairHyperM, dRepairGA and GA+Repair. Our detailed analysis (not shown) indicates that the high *baseline-error-ratio* score of dRepairRIGA and the upward curves of dRepairHyperM, dRepairGA and GA+Repair in the left plot are largely due to the exceptionally low error of these algorithms in one single problem, the G24_3f problem. When solving this problem dRepairRIGA has very low error at all crossover rates and dRepairHyperM, dRepairGA, GA+Repair have very low

error when the crossover rate becomes higher.

Second, the newly proposed algorithms have better overall scores than tested existing DO and CH algorithms in all cases regardless of the crossover rate, except the case of dRepairGA at the extreme crossover rate of 1.0 where it has slightly worse performance than RIGA.

Third, the higher the crossover rate, the worse the performance of repair-based algorithms like GA+Repair, dRepairGA, dRepairRIGA and dRepairHyperM (these algorithms perform best in cases where there is no crossover!). It seems that the use of the normal arithmetic crossover is not suitable for repair-based algorithms. Further analysis is needed to find out the exact reason, but I suspect that, in solving DCOPs the arithmetic crossover operator might reduce the benefit of the repair operator in tracking the moving feasible regions. The repair operator, as described in step 2a of Algorithm 9 (page 151), can be considered a special crossover operator because it is very similar to the normal arithmetic crossover except two differences. First, while in the crossover operator both parents are from the search population, in the repair operator one of the parent is from the search individual and the other is from the reference population. Second, in the repair operator an offspring is only accepted if it is feasible. Due to these two differences, in the repair operator the search outcome is biased so that individuals are constantly pushed toward the feasible regions, and hence in DCOPs where the feasible regions move, the population is able to track those movements. Such a bias cannot be provided by the original arithmetic crossover operator. As a result, when the repair operator is combined with the crossover operator, I suspect that the effect of tracking moving feasible regions might be reduced or even be eliminated. The figure also shows that such algorithms with higher diversity as dRepairRIGA and dRepairHyperM are less affected by the increase of crossover rate. One possible reason might be that when the repair-operator becomes less effective in tracking moving feasible regions due to the crossover operator, the higher diversity brought by the incorporated RIGA and HyperM mechanisms might help alleviating this drawback. In particular, because the RIGA/HyperM mechanisms send more diversified individuals to explore the search space, some individuals will have the chance to be sent to the moving feasible regions, and hence improving the process of tracking this moving region.

Fourth, up to a rate of 0.4, the higher the crossover rate the better the performance of GA, HyperM and RIGA. The performance of these algorithms reach the peak at the rate of 0.4. Beyond this rate the performances decrease but still remain relatively good. The result show

that a crossover rate greater than or equal to 0.1 might be a good choice for existing GA-based DO algorithms to solve DCOPs. The result also shows that compared to HyperM and GA, RIGA is much less affected by the variation of the crossover rate and hence it is able to maintain a relatively high performance regardless of the crossover rate. This result implies that compared to other existing DO strategies, diversity-maintaining strategies like RIGA might be a better choice in solving DCOPs thanks to its robustness under different crossover rates.

6.4.3 Hyper-mutation rate and random-immigrant rate

The second type of parameters to be analysed are the important hyper-mutation and random-immigrant rates used for diversity maintaining/introducing in dynamic optimisation: the hyper-mutation rate (P_H) and the random-immigrant rate (P_R). In this analysis different values of P_H and P_R will be evaluated to see how they would affect the performance of the tested algorithms.

I chose algorithms that use the random-immigrant (RIGA/dRepairRIGA) and hyper-mutation (HyperM/dRepairHyperM) mechanisms for this test. In addition, because HyperM at $P_H = 1.0$ is equivalent to restart GA with elitism and RIGA at $P_R = 1.0$ is equivalent to random search with elitism, in this experiment we will also be able to compare the performance of the tested algorithms with random search and restart GA. It is also noteworthy that even in case P_H and P_R are equal to 1, dRepairRIGA and dRepairHyperM are still not equivalent to random or restart search because of two reasons. First, due to the way dRepairGA-based algorithms works, the search population is not entirely random. This is because every randomised/re-initialised individuals will have to be repaired (and hence are no longer random) before being added to the search population. Second, the reference population is also not entirely randomised or restarted whenever we apply $P_H = 1$ or $P_R = 1$ to the dRepair-based algorithms. This is firstly because the reference population still remains unevolved during each 100-evaluation period and secondly because when the reference population evolves, random individuals are only accepted if they are feasible and better than their parents (see step 3b in Algorithm 8 (page 150)).

In Figure 6.12 I present the results of the algorithms using the default parameters values set out in Table 6.1 (page 178). For this experiment I also tested the algorithms under different crossover rates (from 0.0 to 1.0) and different base mutation values (from 0.0 up to the chosen hyper-mutation/random-immigrant values), but these results are not shown because the behaviours of the algorithms are roughly the same as in Figure 6.12.

Among the observations we got from Figure 6.12, the following are most interesting. First, in order to achieve the best performance the tested algorithms need to use very high mutation rates, which are equivalent to very high levels of diversity. Figure 6.12 shows that algorithms performance increase in accordance with the increase of P_R or P_H until they reach their peaks at the rate of 0.8-1.0. It is also worth noting that the P_H rate where HyperM reaches its peak performance slightly varies depend on the value of the crossover rate. Our detailed analyses (not shown) indicate that the higher the crossover rate, the lower the hyper-mutation rate needed to reach peak performance. For example, at the crossover rate from 0.1 to 0.4, HyperM reaches its peak at $P_H = 1.0$ or $P_H = 0.8$, but at the higher crossover rate from 0.8 to 1.0, HyperM reaches its peak performance at a lower mutation rate $P_H = 0.6$. However, in any case, the results still suggest that high P_H and P_R are necessary to solve the tested problems more effectively.

Surprisingly, even with the highest possible diversity levels, the experimental results show that algorithms such as RIGA and HyperM are still worse or not much better than restart GA and random search. By comparing the performance of HyperM and RIGA at lower P_R and P_H rates with their performance at $P_H = 1.0$ (equivalent to restart GA with elitism and change-detection) and $P_R = 1.0$ (equivalent to random search with elitism) in subplots b and d of Figure 6.12, we can see that HyperM is only able to perform slightly better than restart GA (plus elitism and change-detection) at $P_H = 0.8$ and RIGA is not even able to perform better than random search (with elitism)!.

This observation suggests that, for existing DO algorithms like HyperM and RIGA, the levels of change severity in the search spaces are so large that the DO techniques such as random-immigrant or hyper-mutation seem not to bring much benefit to the algorithms. In other words, toward RIGA and HyperM, there seems to be not much correlation between the environments before and after a change in the tested DCOPs.

The suggestion that changes in the tested problems are very large, however, seems to be contradicted with the actual level of change (medium) that I have set up for each dynamic element in the tested problems (see Table 6.1, page 178). In the experiments, the objective function and constraint functions only change moderately, meaning that there should be a considerable amount of correlation in each function before and after each change, and hence it should be possible to track the changes of objective function or constraint functions if they are solved separately using RIGA and HyperM.

This contradiction leads to the second interesting observation: I have found that the reason for such a contradiction to exist is that although individually the objective function and constraint functions only change gradually, when they are combined to create a DCOP, they can create much more severe changes in the combined constrained search space. One simple example can be seen in the problem G24_4 illustrated in Figure 5.1, page 106. In that problem, if we view the objective function and the constraint functions separately, we can see that both of them change only moderately after each change step (the objective function gradually rotates and the constrained areas gradually move). However, the constrained landscape in the figure shows that the combination of these two types of small changes creates a much more sudden behaviour in the search space: the global optimum moves from one disconnected region to another.

The fact that the small changes in each dynamic element can create larger, much more sudden changes in the combined constrained search space is the reason why we see that the tested algorithms need very high P_H and P_R to get the best overall scores in solving the tested DCOPs. Our detailed analysis (not fully shown due to lack of space) on pairs of problems from the G24 set where one problem is unconstrained dynamic (dF+noC) and the other is constrained dynamic (dF+fC or dF+dC), we always see that the tested algorithms need much higher P_R or P_H rates to get the best results in the constrained dynamic cases (dF+fC or dF+dC). This result prove that the combination of constraints and objective function might make changes in DCOPs much more severe than changes in unconstrained problems. Figure 6.13 provides an illustrated example: the pair G24_8a (dF+noC, dynamic unconstrained) vs G24_8b (dF+fC, dynamic constrained). As can be seen in the figure, to achieve their best performance the tested algorithms need much smaller P_R and P_H rates (0.4-0.6) in the dynamic unconstrained case (dF+noC) than in the dynamic constrained case (dF+fC) (where the required rates are from 0.8 to 1.0).

The fact that RIGA and HyperM perform no better or worse than random search and restart GA, however, does not mean that there is no way to do better in solving DCOPs. This fact is represented in our third observations from Figure 6.12. The figure shows that the newly proposed algorithms - dRepairRIGA and dRepairHyperM - perform much better than both RIGA/HyperM and restart GA/random-search, even when dRepairRIGA and dRepairHyperM use their lowest possible P_R and P_H rates. There are two reasons for the newly proposed algorithms to always perform better regardless of the mutation rates. First, as already analysed

in Subsection 6.3.1, dRepairGA-based algorithms utilise diversified individuals better than the existing DO algorithms as GA/RIGA/HyperM and their corresponding restart/random-search versions. While the aforementioned existing DO algorithms reject many diversified individuals because they are infeasible, the newly proposed algorithms accept both feasible and infeasible solutions, and hence are able to retain more diversified individuals to deal with environmental dynamics. This is also the reason why although dRepairGA-based algorithms generate 20% fewer diversified individuals than GA-based algorithms at the same P_R or P_H rate (see paragraph 6 of Subsection 6.1.4 for explanation), they still reach their peak performance at the same or lower mutation rates than GA-based algorithms. Utilising diversified individuals more effectively is also the reason why dRepairRIGA/dRepairHyperM are less affected by the variation of P_R and P_H rates than RIGA/HyperM, as can be seen from Figure 6.12.

The second, and more important reason for the better performance of dRepairGA-based algorithms is that, while the existing DO algorithms consider the search space as a black box with no available insight knowledge, dRepairGA-based algorithms uses their knowledge about the objective function and constraint to handle dynamic objective function and dynamic constraints differently. As discussed earlier, the sudden changes in the tested DCOPs are composed of less severe elementary changes from the objective functions and constraints. If we only consider the search space as a black box and try to deal with the compositional changes as a whole using existing unconstrained DO techniques, then it is indeed difficult to do better than restart and random-search because the changes are very severe and because as analysed in Chapter 5 it is difficult to combine the goals of DO and CH effectively to satisfy the requirements mentioned in Subsection 5.5.2. However, if we try to deal with objective changes and constraint changes differently, e.g. tracking the moving unconstrained optima and tracking the moving feasible regions separately, it might be possible to do better than restart and random search because firstly elementary changes are usually less severe and secondly we can satisfy the goals of DO and CH separately without any conflict. The fact that objective changes and constraint changes might need to be handled differently is also reflected in our list of suggested requirements for algorithms to solve DCOPs effectively (see Subsection 5.5.2).

As recalled in Subsection 6.1.3, the dRepairGA-based algorithms follow exactly this approach of dealing with elementary changes differently and separately. First, these new algorithms handle changes in the objective function separately by firstly detecting changes based on drops

in objective values and then use RIGA/HyperM mutation to deal with the moving/appearing optima. Second, these new algorithms also handle constraint changes separately by monitoring the boundaries of feasible regions to detect movements and then track the possible movements using the repair operator.

In sum, the results suggest that to solve the tested DCOPs effectively, high mutation/random-immigrant rates are preferable because the levels of change severity are very high. The results also suggest that, if an algorithm attempts to solve the tested DCOPs as black-box problems using existing DO techniques, that algorithm would not be able to perform better than the simple restart and random approaches because the changes are very severe. However, if we follow a "divide-and-conquer" approach to firstly decompose changes into objective changes and constraint changes and then handle the elementary changes differently using different techniques, we will be able to solve the problems better because the elementary changes are generally less severe. The efficiency of this "divide-and-conquer" approach is proved by the significantly better overall performance of dRepairGA-based algorithms compare to existing DO and CH algorithms, regardless of the chosen parameter values.

6.4.4 Base mutation rate

The third parameter that we are going to analyse is the *base mutation rate* (P_b). Figure 6.14 shows our analysis of how changing the base mutation rate would affect the performance of the tested algorithms when all other parameters are kept constant. This analysis will help to answer two questions. The first question is about the effect of P_b on the performance of GA-based and repair-based algorithms in solving DCOPs. The second question is about the effect and interaction between the base mutation and the hyper-mutation rate in HyperM/dRepairHyperM and the interaction between the base mutation and the random-immigrant rate in RIGA/dRepairRIGA. We can also compare the difference between HyperM/ dRepairHyperM, RIGA/ dRepairRIGA and GA/ dRepairGA when the base mutation P_b changes.

The algorithms selected for this analysis are GA, dRepairGA, HyperM, dRepairHyperM, RIGA and dRepairRIGA. Algorithms like Genocop and dGenocop were not tested because, similar to the case of crossover, due to the special design in Genocop III/dGenocop changing the mutation rate P_b would affect the rate of nine other specialised operators. In that case, any impact in performance might be caused by not only the change in mutation rate but also by

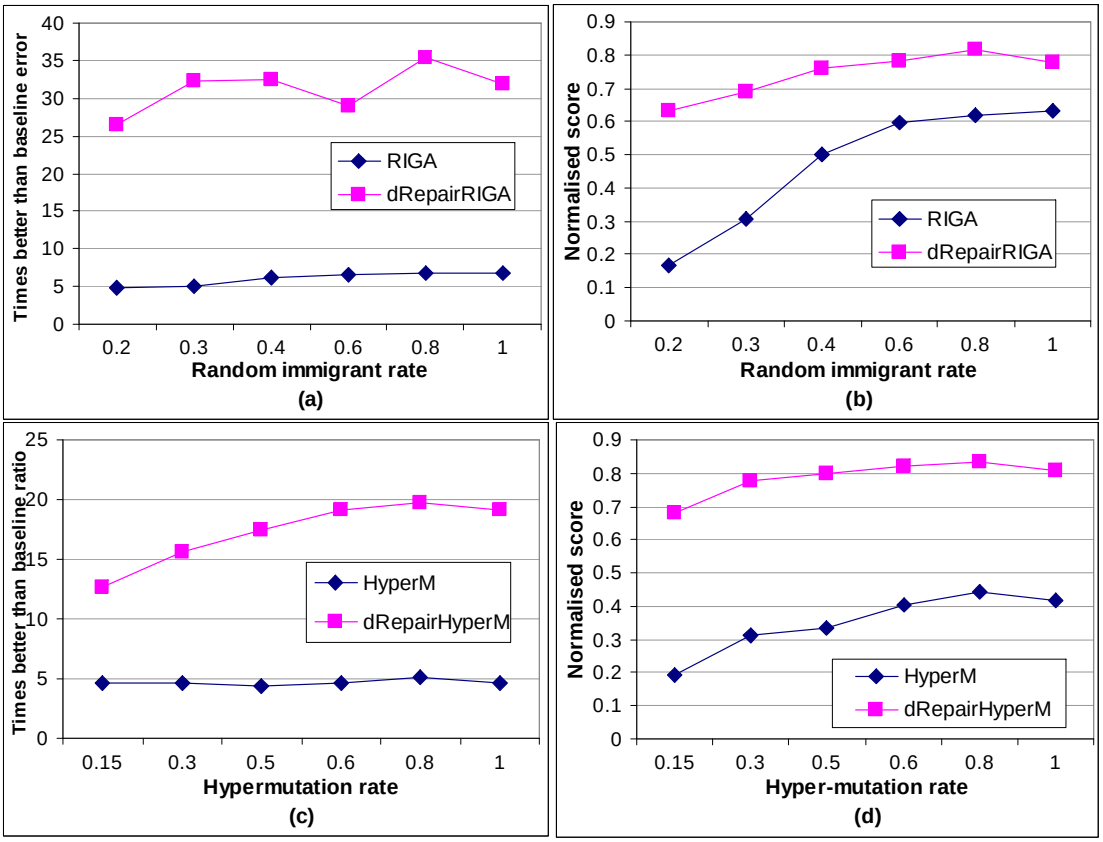

Figure 6.12: This figure shows an analysis on how changing the random-immigrant rate (subplots a and b) and the hyper-mutation rate (subplots c and d) would affect the overall performance of RIGA/dRepairRIGA and HyperM/dRepairHyperM on all the 18 tested problems. The impact of different hyper-mutation rates and random-immigrant rates on algorithm performance is calculated using two performance measures: the *baseline-error-ratio* on the left-hand side and the *normalised score* (see Equation 6.3) on the right-hand side. The higher the y-axis values, the better the performance.

the consequent changes in the probability rates of other specialised operators. At the extreme value $P_b = 1.0$, the GA-based algorithms (GA/RIGA/HyperM) become random search with elitism (dRepair-based algorithms, however, are not equivalent to random search at this rate as already mentioned in Subsection 6.4.3). Because of that fact in this experiment we will be able to see how the existing algorithms and new algorithms perform compared to random search. In this experiment I will also compare the tested algorithms with restart GA. Of course it is not totally fair to compare the tested algorithms with restart GA because the former have to spend computational cost to detect/adapt with changes automatically while in the latter we assume that changes are known and hence we can restart the algorithm accordingly. In addition, the fact that restart GA cannot reuse previous solutions might be a big disadvantage in many practical

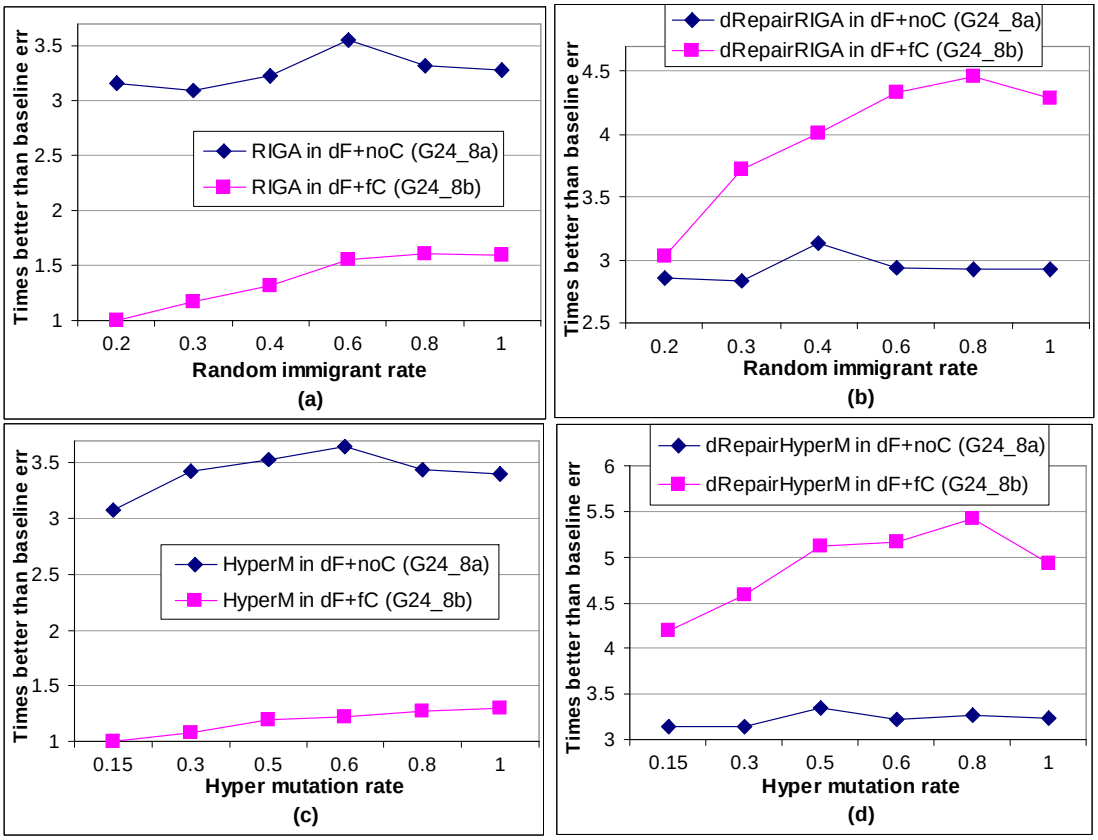

Figure 6.13: This figure shows the difference in the best random-immigrant/hyper-mutation rates that the tested algorithms needed in the dynamic constrained case (G24_8b dF+fC), compared to the dynamic unconstrained case (G24_8a dF+noC). We can see that the tested algorithms need much smaller mutation rates (0.4-0.6) in the dF+noC case than in the dF+fC case (where the required rates are from 0.8 to 1.0) to achieve the best performance.

situations. However, the comparison will tell us how useful the algorithms are compared to restart GA when changes are known and the users do not need to re-use existing solutions.

The result in Figure 6.14 shows that the algorithms have the same general behaviours, which is as expected: algorithm performance is significantly worse at the smallest mutation rate P_b, then the performance increases when P_b increases until the performance reaches its peak at mid-range P_b values, and finally the performance decreases again (but only less slightly) when P_b reaches the extreme high values ($P_b > 0.7$). At the extreme $P_b = 1.0$ all GA-based algorithms have the same normalised score and all repair-based algorithms have the same normalised score.

More detailed analysis on the result in Figure 6.14 reveals some interesting observations as follows.

First, the top normalised scores are gained at high mutation rates (larger than 0.2), con-

firming our previous finding that the overall levels of change severity in the tested problems are high.

Second, the result shows that, similar to the case of testing random-immigrant/hyper-mutation rates in the previous subsection, GA/RIGA/HM also do not perform significantly better (and even worse in certain cases) than random search and restart GA. This ineffectiveness of GA-based algorithms is also due to the fact that the tested DCOPs have high levels of change severity although their elementary changes in objective function and constraints are only moderate. Similar to the previous experiments in Subsection VI-C, our detailed analysis (not fully shown due to the lack of space) on pairs of problems from the G24 set where one problem is unconstrained dynamic (dF+noC) and the other is constrained dynamic (dF+fC or dF+dC) also shows that the tested algorithms need much higher P_b to get the best results in the constrained dynamic cases (dF+fC or dF+dC) than in the unconstrained dynamic case (dF+noC). An example, again on the pair G24_8a (dF+noC, dynamic unconstrained) vs G24_8b (dF+fC, dynamic constrained), is given in Figure 6.15 . As can be seen in the figure, to achieve their best performance the tested algorithms need smaller P_b rates (0.5-0.6) in the dynamic unconstrained case (dF+noC) than in the dynamic constrained case (dF+fC) (where the required rates are 0.8 to 0.9).

Third, the fact that GA/RIGA/HM are not able to perform much better than restartGA and random search does not mean that there is no way to do better. Figure 6.14 shows that new repair-based algorithms always perform significantly better than existing algorithms given the same base mutation rate. Algorithms as dRepairHyperM and dRepairRIGA even perform better than existing algorithms, including random search and restart GA, at any base mutation rate. As discussed previously in Subsection 6.4.3, the better performance of dRepair-based algorithms might be due to their "divide-and-conquer" approach to handle changes in objective functions and constraints separately and differently.

Some other observations from the results are (1) The impact of base mutation become less significant from the base mutation rate of 0.1; (2) Algorithms with diversity-maintaining strategies (RIGA/HyperM/dRepairRIGA/dRepairHyperM) are more robust to deal with the variation in base mutation rate and (3) Given an unknown problem dRepairRIGA should be the one to be chosen for solving the problem. The result in Figure 6.14 shows that this algorithm has both the highest overall normalised score (which proves that it is robust) and the highest overall

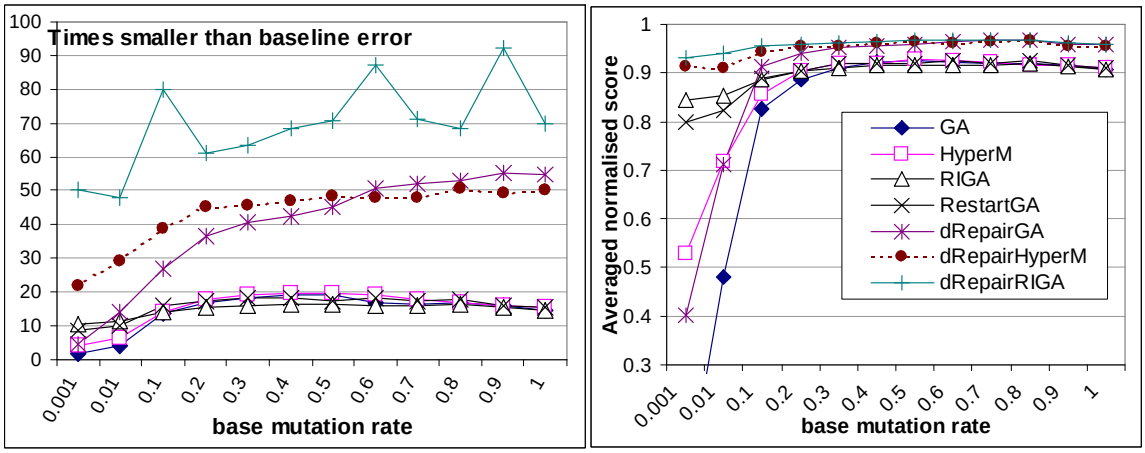

Figure 6.14: This figure shows an analysis of how changing the base mutation rate would affect the overall performance of the tested algorithms on all the 18 tested problems. The impact of different base mutation rates on algorithm performance is calculated using two performance measures: the *baseline-error-ratio* on the left-hand side and the *normalised score* (see Equation 6.3) on the right-hand side. The higher the y-axis values, the better the performance.

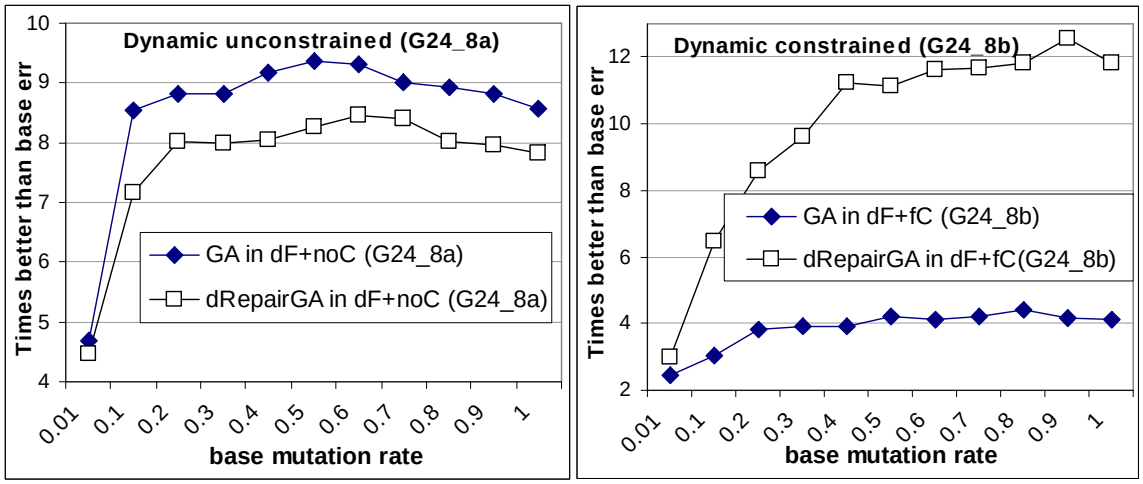

Figure 6.15: This figure shows the difference in the best base-mutation rates that the tested algorithms needed to solve a dynamic constrained problem (G24_8b dF+fC), compared to the dynamic unconstrained case (G24_8a dF+noC). To avoid the graph being too cluttered, we only included GA and dRepairGA, but the behaviours of other tested algorithms are similar.

baseline-error-ratio (which proves that it can gets highly precise results).

6.4.5 Replacement ratio and the effect of Lamarckian evolution

One of the important parameters for repair-based algorithms in solving static constrained problems is the *replacement ratio*. This parameter determines the percentage of *repaired individuals*

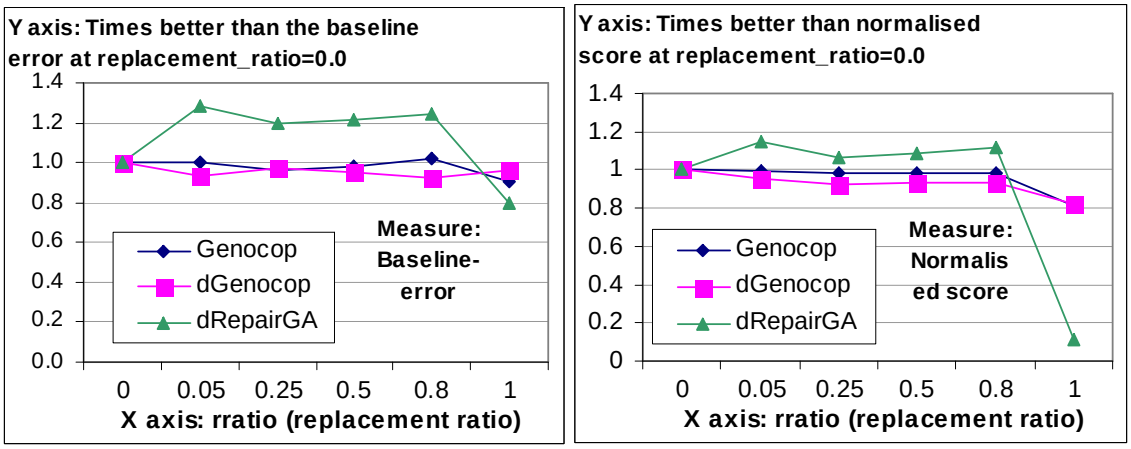

Figure 6.16: This figure shows how changing the replacement rate would affect the performance of the tested algorithms. The impact of different replacement rates is evaluated using two measures: the *baseline-error-ratio* on the left-hand side and the *normalised score* (see Equation 6.3) on the right-hand side. The calculation is done as follows. First, the offline-error (left) and the normalised-score (right) of algorithms with no replacement (Baldwinian, replacement rate=0.0) are recorded as the baseline error/score. Then on the left-side figure we calculate the ratio between the baseline-errors and the errors of each algorithm using differrent replacement rates. On the right-side figure we calculate the ratio between the baseline normalised-score and the normalised-scores of each algorithm using differrent replacement rates. These two ratios are repesented in the vertical axes of the two figures on the left and right sides. The purpose is to see if changing the replacement rates can make the algorithm perform better (y-axis value > 1) or worse (y-axis value < 1).

(the individual **z** in the routine *Repair*, Algorithm 9, page 151) to replace individuals in the population. Any non-zero value of this parameter means that the algorithm follows a Lamarckian-evolution approach because the outcome of learning (repaired individuals in this case) can be applied directly to the evolution process by changing the chromosomes of individuals. In case the value is zero, i.e. no repaired individual is used to replace the original individuals, the algorithm follows a Baldwinian-evolution approach because learning can only affects evolution indirectly.

In this subsection I will investigate if Lamarckian evolution could bring any benefit to the tested algorithms in solving the problems in the G24 benchmark set, and if yes, what would be the most suitable replacement ratio. For this experiment I chose the original Genocop III, the newly proposed dGenocop and dRepairGA as test algorithms.

Figure 6.16 shows our analysis of how changing the replacement ratio would affect the performance of repair-based algorithms when all other parameters are kept constant.

The experimental results show that Lamarckian evolution has little effect on the performance

of Genocop III and our dynamic constrained optimisation version dGenocop (the performance was decreased by up to 7%), except that the performance of these two algorithms drop when the replacement ratio becomes 100%.

Lamarckian learning, however, has positive effect on the performance of our GA-based repair algorithms such as dRepairGA/dRepairRIGA/dRepairHyperM. Although in the previous experiments we set the replacement ratio to zero (and hence disabled Lamarckian learning to maintain a fair comparison with GA+Repair), Figure 6.16 shows that if we maintain a replacement ratio from 0.05 to 0.8, the performance of dRepairGA can be improved by 20-28% (baseline error) or 7-15% (normalised score) of which the biggest improvement can be gained with a replacement ratio of 0.05. The fact that a replacement ratio of 0.05 (5%) achieves the best performance also conforms with the 5% heuristic rule suggested by Orvosh and Davis (Orvosh & Davis 1993) for some static combinatorial problems.

The result also shows that Lamarckian evolution can also provide negative effect if overused. As can be seen in Figure 6.16, when 100% of original individuals are replaced by the repaired individuals, the performance of all three algorithms, especially dRepairGA, can be decreased significantly.

6.5 Summary

6.5.1 Summary of contributions and findings

This chapter has made the following contributions.

1. Propose a new approach to solving DCOPs: combining existing DO techniques with CH techniques to handle objective-function changes and constraint-function changes separately and differently. The approach was applied to two EAs: GA and Genocop III to create new algorithms named dRepairGA, dGenocop and variants (with better results than the tested existing DO and CH algorithms in all groups of problems except the static cases).

   (a) Modified an existing CH technique (the repair method) to track the moving constraints and combined it with existing DO techniques (RIGA and HyperM) to handle objective-function changes.

   (b) Used different techniques to detect objective-function changes and constraint-function changes separately and differently.

2. Offer a deeper understanding of the behaviours and characteristics of DCOPs. This include the following interesting findings

   (a) Changes in DCOPs are usually more sudden than the unconstrained case. Such sudden changes might make existing DO algorithms like RIGA and HyperM perform worse than restart and random search in solving DCOPs.

   (b) In DCOPs the presence of (dynamic) constraints might not always make the problems more difficult (than the unconstrained case) as intuitively expected. For the newly proposed algorithms, the presence of constraints actually makes some of the tested problems easier to solve. Also for the new algorithms, the presence of multiple disconnected feasible regions might make some of the tested DCOPs easier to solve (the larger the barriers the easier).

3. Give an insight into how different algorithmic components influence algorithm performance in DCOPs:

   (a) Analyse the impact of evolutionary operators (crossover, base mutation, hyper-mutation, random-immigrant), Lamarckian/Baldwinian learning, and the newly proposed routines DetectChange, UpdateSearchPop, UpdateReferencePop and OOR.

   (b) Provide a guideline for choosing the ideal parameter settings for each algorithm to solve DCOPs.

4. Propose two new performance measures for analysing the performance of algorithms in DCOPs.

   (a) The *optimal region tracking* score to evaluate the ability of algorithms to track the moving feasible regions.

   (b) The *normalised score*, to quantitatively evaluate the overall performance of algorithms in groups of different problems in an unbiased way.

6.5.2 Advantages of the proposed methods

The first advantage of the proposed algorithms is that they are able to overcome the drawbacks of existing DO and CH strategies in solving DCOPs. This advantage makes the proposed

algorithms work significantly better than the tested existing algorithms in the tested DCOPs, regardless of the used parameter values.

The second advantage of the proposed algorithms is that they are robust. The experimental results show that, among the tested problems the new algorithms are able to work well not only in DCOPs but also in most other types of the tested problems except in the group of static problems.

The third advantage of the proposed methods is their generality. The methods can be hybridised with any population-based continuous EA. In this chapter I have integrated them with GA and Genocop III, both with good results.

The fourth advantage of the proposed methods is the ability to work without the need of choosing many parameters. The only mandatory parameters are the population size and the mutation/hyper-mutation rate or random-immigrant rate. Optional parameters are maximum number of detectors and replacement ratio. There is also a crossover rate parameter but our detailed analysis suggests that for the newly proposed algorithms no crossover would give the best results.

6.5.3 Shortcomings of the proposed methods

The biggest disadvantage of the proposed methods, which is also the disadvantage of all repair-based methods, is that they require a considerable number of feasibility checkings: during the repair process of an individual, the constraint functions might be evaluated many times until a feasible individual is found or until the number of iterations reaches a given limit. In addition, the DetectChange routine also evaluates the constraint functions of detectors in every generation, adding more cost to the total number of constraint evaluations. Due to this reason, the proposed methods, and other repair-based methods, might not be suitable for solving problems with very expensive constraint functions and problems with very small feasible areas (because the repair-based methods would need to take a lot of feasibility checkings to find a feasible solution). Adaptive method as in equation 6.1 can be used to give a balance between the number of constraint function evaluations and computational cost, but the performance might be affected.

Another limitation is the ability of the proposed methods to detect objective changes that increase the best fitness values. Because the hyper-mutation strategy used in the proposed methods only relies on performance drops to detect changes, it will not be able to react to

changes that "increases" the performance.

Chapter 7

Dynamic time-linkage optimisation

This chapter studies some unknown characteristics and the solvability of some classes of dynamic time-linkage problems (DTPs). As we have seen in the review in Subsection 3.3.2 DTP is a common type of dynamic optimization problems (DOPs) in both real-world combinatorial and continuous domains but has not yet received enough attention from the Evolutionary Computation research. They are defined as problems where "...there exists at least one time $0 \leqslant t \leqslant t^{end}$ for which the dynamic optimization value at time t is dependent on at least one earlier solution..." (Bosman 2007).

Although the importance of DTPs have been shown through their presence in a broad range of real-world applications, due to the lack of research attention there are still many characteristics that we do not fully know about this type of problem. For example, how should we define and classify DTPs in detail; is there any characteristics of DTPs that we do not know; with these characteristics are DTPs still solvable; and what is the appropriate strategy to solve them. Chapter 4 has already addressed the first issue: providing a formal definition for DTPs and DOPs in general. Here in this chapter the other issues will be partially addressed. First, although it is believed that DTPs can be solved to optimality with a perfect prediction method to predict future function values (Bosman 2005, Bosman & Poutré 2007), in this chapter I will discuss a new class of DTPs where it might not be possible to solve the time-linkage problems to optimality because there is not always the possibility to perfectly predict the future. In addition, in this type of DTPs if we try to predict the future based on information from the past, we may even get worse results than not using a predictor at all. I will then carry out some experiments to verify the finding and will also discuss under which situation can we solve this particular type

of DTPs.

7.1 Time-deceptive and the anticipation approach

According to (Bosman 2007), a dynamic problem is said to be time-deceptive toward an optimiser if the problem is time-linkage and the optimiser cannot efficiently take into account this time-linkage feature during its optimization process.

Bosman(Bosman 2005) illustrates this property by proposing the following test problem:

$$\text{given } n = 1; h(x) = e^x - 1; \qquad \max_{x(t)} \left\{ \int_0^{t^{end}} f\left(x\left(t\right), t\right) dt \right\} \tag{7.1}$$

$$f(x_t, t) = \begin{cases} -\sum_{i=1}^{n} \left(x\left(t\right)_i - t\right)^2 & \text{if } 0 \leqslant t < 1 \\ -\sum_{i=1}^{n} \left[\left(x\left(t\right)_i - t\right)^2 + h\left(\left|x\left(t-1\right)_i\right|\right)\right] & \text{otherwise} \end{cases}$$

The benchmark problem above is a DTP because for any $t \geqslant 1$, the current value of $f\left(x, t\right)$ depends on $x\left(t-1\right)$ found at the previous time step.

An interesting property is revealed when we try to optimise the above problem using the traditional approach: optimizing the present. That property is: *the trajectory formed by optimum solutions at each time step might not be the optimal trajectory.* For example, in figure 7.1 we can see that the trajectory of $f(x^*, t)$ when we optimise the present (with optimum solution $\mathbf{x}^*\left(t\right) = t$ at the time step t) is actually worse than the trajectory of a $f(x^0, t)$ with a simple solution $\mathbf{x}^0 = 0\ \forall t$. It means that the problem is deceptive because an optimiser following the traditional approach is not able to take into account the time-linkage feature.

Bosman (Bosman 2005, Bosman 2007) suggested that DTPs can be solved to optimality by estimating the values of the function for future times given a trajectory $\cup_{t=0}^{t^{now}} \{f_t, t\}$ of history data and other previously evaluated solutions. From that estimation, we can choose a future trajectory with optimal future function values. In other words, it is suggested that time-linkage problems can be "*solved to optimality*" by prediction methods and the result could be "*abitrarily good*" if we have a "*perfect predictor*"(Bosman 2005, Bosman & Poutré 2007, Bosman 2007) [1]. The authors also made some experiments on the test problem mentioned in eq. 7.1 and on the dynamic pickup problem, showing that under certain circumstances prediction methods do help

[1] A predictor, as defined in (Bosman 2007, line 8-12, pg 139), is "*a learning algorithm that approximates either the optimization function directly or several of its parameters... When called upon, the predictor returns either the predicted function value directly or predicted values for parameters*". Hence, perfect predictors should be ones that can predict values exactly as the targets.

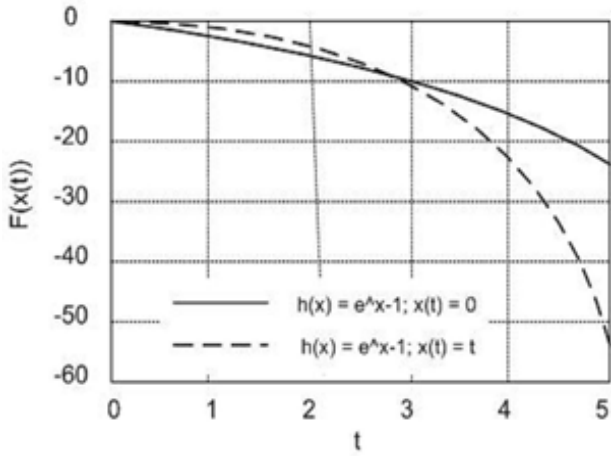

Figure 7.1: This figure (reproduced from (Bosman 2005)) illustrates the time-deception property. We can see that the trajectory of $f(x_t)$ when we optimize the present (dash line, with optimum solution $x(t) = t$) is actually worse than the trajectory of $f(x_t)$ with a simple solution $x(t) = 0$ (the solid line). To solve this problem to optimality, we need to use a predictor to predict the trajectory of function values given different outcomes of current solutions, then choose the one that give us the maximum profit in the future.

to improve the performance of the tested algorithms.

7.2 Can anticipation approaches solve all DTPs?

Contrary to existing belief, I will show below that there might be cases where the hypothesis above does not hold: if during the predicted time span, the trajectory of the future function values changes its function form, it might not be possible to solve the time-linkage problems to optimality because there is not always the possibility to perfectly predict the future.

Let us consider the situation where predictors help achieving optimal results first. At the current time $t^{now} \geqslant 1$, in order to predict the values of $f(x(t))$ at a future time t^{pred}, a predictor needs to take the history data, for example the previous trajectory of function values $Z^{[0,t^{now}-1]} = \cup_{t=0}^{t^{now}-1} \{f_t, t\}$, as its input. Given that input, a perfect predictor would be able to approximate correctly the function form of $Z^{[0,t^{now}-1]}$ and hence would be able to predict precisely the future trajectory $Z^{[t^{now},t^{pred}]}$ if it has the same function form as $Z^{[0,t^{now}-1]}$.

One example where predictors work is the problem in eq. 7.1. In that problem, for each trajectory of $x(t)$ the trajectory of $f(x(t))$ always remains the same. For example with $x(t) = t$ the trajectory is always $1 - e^{t-1}$ or with $x(t) = 0$ the trajectory is always $-t^2$ (see figure 7.1). As a result, that problem is predictable.

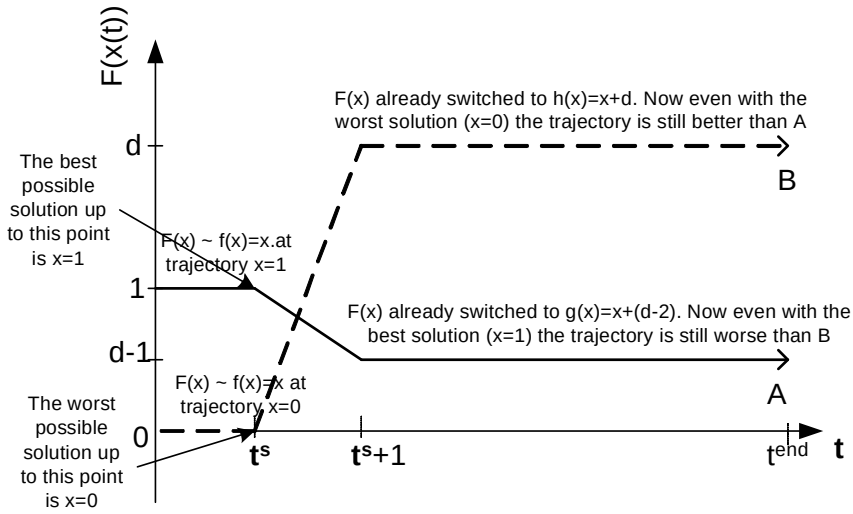

Figure 7.2: This figure illustrates a situation where even the best predictor + the best algorithm (A) still perform worse than the worst predictor + the worst algorithm (B) due to the prediction-deceptive property of the problem in eq.7.2. Assume that we want to predict the trajectory of $F(x)$ from $[0, t^{end}]$. In case A, the best predictor allows us to predict $F(x) \sim f(x) = x$ in just only one time step $[0, 1]$. With that perfect prediction the algorithm is able to find the best solution $x = 1$, which is valid until t^s. Now at t^s although the history data tells the predictor that the trajectory must still be $F(x) \sim f(x) = x$, according to eq.7.3 the actual $F(x)$ does switch to $g(x) = x + (d - 2)$, which is the worst trajectory. In other words, the best predictor chose the worst trajectory to follow. On the contrary, in the case B the worst predictor+worst algorithm actually get benefit from the switch: the terrible solution $(x = 0)$ they found during $[0, t^s]$ does help them to switch to $F(x) \sim h(x) = d + x$, whose trajectory after t^s is always better than A regardless of the value of x.

Now let us consider a different situation. If at any particular time step $t^s \in \left[t^{now}, t^{pred}\right]$, the function form of $Z^{[t^{now}, t^{pred}]}$ changes, the predicted trajectory made at t^{now} to predict $f\left(x\left(t\right)\right)$ at t^{pred} is no longer correct. This is because before t^s there is no information about how the the function form of $Z^{[t^{now}, t^{pred}]}$ would change. Without such information, it is impossible to predict the optimal trajectory of function values after the switch, regardless of how good the predictor is. It means that the problem cannot be solved to optimality because it is not possible to perfectly predict the future.

To illustrate this situation, let us consider the following simple problem where the trajectory of function values changes over time (illustrated in figure 7.2).

$$\widehat{F}\left(x_t\right) = a_t f\left(x_t\right) + b_t g\left(x_t\right) + c_t h\left(x_t\right) \qquad 0 \leqslant x_t \leqslant 1 \tag{7.2}$$

where $\widehat{F}(x)$ is the full-description form[2] of a dynamic function; $f(x_t) = x_t$; $g(x_t) = x_t + (d-2)$; $h(x_t) = x_t + d$; a_t, b_t and c_t are the time-dependent parameters of $\widehat{F}(x_t)$. Their dynamic drivers are set out as follows:

$$\begin{cases} a_t = 1; b_t = c_t = 0 & \text{if } (t < t^s) \\ a_t = 0; b_t = 1; c_t = 0 & \text{if } (t \geqslant t^s) \text{ and } \left(\widehat{F}_{t^s-1}\left(x^G_{t^s-1}\right) \geqslant 1\right) \\ a_t = 0; b_t = 0; c_t = 1 & \text{if } (t \geqslant t^s) \text{ and } \left(\widehat{F}_{t^s-1}\left(x^G_{t^s-1}\right) < 1\right) \end{cases} \tag{7.3}$$

where $t^s > 1$ is a pre-defined time step, $d \in \mathbb{R}$ is a pre-defined constant, and $x^G_{t^s-1}$ is a single solution produced at $t^s - 1$ by an algorithm G. Eq. 7.3 means that with $t < t^s$, the form of $\widehat{F}(x_t)$ is always equal to $f(x_t)$; with $t \geqslant t^s$, depending on the solution of $x^G_{t^s-1}$ the form of $\widehat{F}(x_t)$ would switch to either $g(x_t)$ or $h(x_t)$.

In the above problem, because at any $t \geqslant t^s$ the values of a_t, b_t and c_t (and consequently the value of the function $\widehat{F}$) depend on the solution found by G at $t^s - 1$, according to the definition in (Bosman 2007) the problem is considered time-linkage.

This problem has a special property: at any $t < t^s$ one can only predict the value of $\widehat{F}$ up to $t^s - 1$. Before t^s, history data does not reveal any clue about the switching rule in eq. 7.3, hence it is impossible to predict (1) whether the function will switch at t^s; (2) which value $x^G_{t^s-1}$ should get to switch $\widehat{F}(x_t)$ to $g(x_t)$ / $h(x_t)$ and (3) which form, g or h, would provide better future trajectory.

Even worse, even a predictor that can perfectly learn the current function form of the system might still be deceived to provide worse result than not using any predictor while solving this time-linkage problem! Figure 7.2 illustrates the situations where the best predictor could provide the worst result while the worst predictor could provide better results after t^s!

Problems like this example, i.e. time-linkage problems with function forms switching from one to another, is very common in real-world systems. One common class of problems with this property is the class of hybrid systems. According to Tafazoli & Sun (2006), hybrid systems are real-life systems that can evolve according to different dynamics at different times. At each time step the behaviour of the system is controlled by only one dynamics (one mode), and then depending on the behaviour of the system, at some point the system may switch from one

[2] The concepts like *full-description forms*, *time-dependent parameters* and *dynamic drivers* have been defined and described in Chapter 4.

dynamics to another (switch mode). Examples vary from simple systems like the bouncing ball (where the state switches from falling to bouncing when it meets the ground and vice versa) to complex systems like the autopilot programs in commercial airplanes (where the airplane automatically switches from one flying mode to another). Our survey of real-world applications also show that about 30% of the surveyed applications in the continuous domain are hybrid systems, and all of them have the time-linkage properties (see Subsection 3.2.2). In these applications, if we solve the problems completely *online* as a black-box without any knowledge, we will not be able to solve them to optimality because it will not be possible to predict how the systems will switch their function forms and how the function forms after the switch will be.

Summarising, the example problem I proposed in this section illustrates a common class of DTPs (but has not been studied by the EC community yet) where it is not guaranteed to get optimal results because it is impossible to find a perfect predictor to predict the function values using history data. We call this class *time-linkage problems with unpredictable optimal function trajectories*. The example illustrates a special case where any predictor that relies on past data can be deceived and hence provide the worse results than not using predictor at certain time steps. We call these types of problems the *prediction-deceptive time-linkage problems*.

In Section 7.4, I will carry out some experiments to demonstrate a prediction-deceptive time-linkage problem and its effect on the performance of an algorithm that predicts the future function values based on history data.

7.3 Solving prediction-deceptive time-linkage problems

Prediction-deceptive DTPs are challenging and only under limited circumstances can we solve them to optimality. The answer of whether we can solve them to optimality or at least to avoid being deceived would depend on whether we have to solve them totally online or partially online, and whether do we have to solve the problem as a complete black box or can we get any problem-specific information.

If we have to solve the problem online as a black box, there is not much thing that we can do. Knowing that the problem is prediction-deceptive, we might try not to use anticipation approaches to avoid being deceived. However, there is no guarantee that other approaches would work better.

In real-world applications, however, it might be possible to solve the problem in a partially

online way and also there might be some problem-specific information available so the problem can be solved as a partial black-box. Our survey of real-world applications in Chapter 3 shows that in most of the surveyed hybrid systems, the problems are not totally black box because the mathematical function forms of the possible switch-modes and the switching rules have already been calculated offline based on observation data from real systems or from simulation e.g. see : (Ahmad & Liu 2008, Houwing *et al.* 2007, Fiacchini *et al.* 2008, Long *et al.* 2007). However, because there are modelling errors or disturbances, these mathematical function forms might not exactly reflect the current status of the actual systems. Because of that, the problems still need to be solved online. During the online phase the actual function form of the system will be learned/predicted based on history data to "correct" any mis-modelling due to errors/disturbances. Another reason for certain real hybrid systems to be solved online even when their mathematical model is known is that the initial state of the system is unknown and hence it is unclear which dynamic mode the system is currently in or what are the correct values of the system's parameters when the system is started. Due to that reason, the problem is also needed to be solved online and during the optimisation process the correct initial state/dynamic mode of the system will be estimated.

In time-linkage problems with function forms switching from one to another and with the knowledge about switching rules like these, I believe that it might still be possible to solve them using prediction method while avoid being deceived. In order to do that, the solver needs to take into account not only the current function value and the future consequent values of the current function forms, but also the consequent function switches and the future values of the new function after a switch has been made.

Specifically, given a time-linkage problem with switching function forms and the knowledge of the switching rules, in order to solve the problem to optimality during the period $\left[t^{now}, t^{end}\right]$, at the current moment t^{now} an algorithms needs to find the solution $x\left(t^{now}\right)$ and a set of switching time $\{T_1, ..., T_{n-1}, T_n\}$ where $T_n = t^{end}$ to optimise the future trajectory and future switches:

$$\text{optimise}\left\{ f\left(x\left(t^{now}\right)\right) + \sum_{t=t^{now}+1}^{T_1} f_{pred}\left(x\left(t\right)\right) + \sum_{T_i=T_1}^{T_{(n-1)}} \sum_{t=T_i+1}^{T_{(i+1)}} f_{switch}\left(x\left(t\right), x\left(T_i\right)\right) \right\} \tag{7.4}$$

where f_{pred} is the estimated function form of the current dynamic model of the system and f_{switch} is the expected function form of the dynamic model that the system will switch into

under the estimated value of $x(T_i)$.

To make sure that the system does switch its mode at the chosen switching times $\{T_1, ..., T_{n-1}, T_n\}$ and does switch to the most suitable modes, the solver needs to produce the right the value of $x(T_i)$, the solution produced at the switching time. In some cases, e.g. (Sonntag *et al.* 2008, Summers & Bewley 2007), the switching times are fixed or pre-determined and hence the solver does not need to determine $\{T_1, ..., T_{n-1}, T_n\}$ but just to estimate the current function form and then choose the switch-modes that will provide the greatest benefits when the switching time comes.

In summary, for time-linkage problems with switching function forms where the knowledge of the switching rules is available, it is possible to solve the problem more effectively if during the optimisation process we take into account not only the current function value and the future consequent values of the current function forms, but also the consequent function switches and the future values of the new function after a switch has been made. In other words, it is possible to solve the problem more effectively if the algorithm optimise the problem using the objective function described in Equation 7.4. In the next section, I will carry out some experiments to verify the efficiency of the method proposed in this section in solving time-linkage prediction-deceptive problems.

7.4 Experiments

In this section I will carry out some experiments to verify

1. the impact of the time-deceptive property in time-linkage problems on optimisation algorithms that follow the optimising-the-present approach;

2. the efficiency of the learning-the-current-function-form approach in solving time-deceptive time-linkage problems;

3. the impact of prediction-deceptive property in time-linkage problems on optimisation algorithms that follow the learning-the-current-function-form approach;

4. the efficiency of our proposed approach in solving prediction-deceptive time-linkage problems when information about switching rules is available

Points (1) and (2) have already been illustrated in (Bosman 2005, Bosman 2007), but here the verification will be re-done again because these results will be needed for verifying points

(3) and (4).

7.4.1 Test problems

Problem DTP1

In (Bosman 2005), a test time-linkage problem with the time-deceptive property has been proposed. This problem will be used in this section to verify the points (1) and (2) above. The test problem has been described in Equation 7.1, page 225). Here I will represent the problem (with $n = 1; h(x) = x^2$) in a slightly different way to make it conform to our definition framework in Chapter 4 and make the change severity adjustable:

$$\max_{x(t)} \left\{ \sum_{0}^{t_{end}} \widehat{F}\left(x_t\right) \right\} \tag{7.5}$$

where

$$\widehat{F}\left(x_t\right) = f^1 = \begin{cases} -\sum_{i=1}^{n} \left(x\left(t\right)_i - s.t\right)^2 & \text{if } 0 \leqslant t < \lfloor 1/s \rfloor \\ -\sum_{i=1}^{n} \left[\left(x\left(t\right)_i - s.t\right)^2 + \left[x\left(t - \lfloor 1/s \rfloor\right)_i\right]^2\right] & \text{otherwise} \end{cases}$$

and $s \in R$ is the change severity, $0 < s \leqslant 1$.

The problem is named DTP1. Experiments on this problem will be presented in Subsection 7.4.3.

Problem DTP2

To verify points (3) and (4), we need to create a problem with the prediction-deceptive property. To maintain continuity and to re-use the results I got from the process of verifying points (1) and (2), I modify the original Bosman problem in Equation 7.5 to make it a prediction-deceptive problem. Particularly, up to the change step t^{switch} the problem is similar to DTP1, but at t^{switch} the problem switches its function form depending on the function value found by the algorithm at t^{switch}. If the found function value is high, the problem switches to a low-value trajectory. Vice versa, if the value found at t^{switch}, the problem switches to a high-value trajectory. Details of the problem are as follow:

$$\max_{x(t)} \left\{ \sum_{0}^{t_{end}} \widehat{F}\left(x_t\right) \right\} \text{ , } \widehat{F}\left(x_t\right) = a_t f^1\left(x_t\right) + b_t f^2\left(x_t\right) + c_t f^3\left(x_t\right) + d_t f^4\left(x_t\right) \tag{7.6}$$

where $\widehat{F}(x)$ is the full-description form[3] of the mathematical descriptions f^1, f^2, f^3, f^4 (given in Equation 7.7); a_t, b_t, c_t, d_t are the time-dependent parameters of $\widehat{F}(x_t)$ (their dynamic drivers are given in Equation 7.8).

Below are the descriptions of f^1 (the original Bosman function), f^2, f^3, and f^4 :

$$\begin{cases} f^1(x_t,t) = \begin{cases} -\sum_{i=1}^{n}\left(x\left(t\right)_i - s.t\right)^2 & \text{if } 0 \leqslant t < \lfloor 1/s \rfloor \\ -\sum_{i=1}^{n}\left[\left(x\left(t\right)_i - s.t\right)^2 + \left[x\left(t - [1/s]\right)_i\right]^2\right] & \text{otherwise} \end{cases} \\ f^2(x_t,t) = -60 \\ f^3(x_t,t) = -40 \\ f^4(x_t,t) = -10 \end{cases} \tag{7.7}$$

where $s \in R$ is the change severity, $0 < s \leqslant 1$.

Below are the descriptions of the dynamic drivers of the time-dependent parameters a_t, b_t, c_t, d_t:

$$\begin{cases} a_t = 1; b_t = c_t = d_t = 0 & \text{if } \left(t \leqslant t^{switch}\right) \\ a_t = 0; b_t = 1; c_t = d_t = 0 & \text{if } \left(t > t^{switch}\right) \text{ and } \left(-36 \leqslant \widehat{F}\left(x_{t^{switch}}\right)\right) \\ a_t = 0; b_t = 0; c_t = 1; d_t = 0 & \text{if } \left(t > t^{switch}\right) \text{ and } \left(-50 \leqslant \widehat{F}\left(x_{t^{switch}}\right) < -36\right) \\ a_t = 0; b_t = 0; c_t = 0; d_t = 1 & \text{if } \left(t > t^{switch}\right) \text{ and } \left(\widehat{F}\left(x_{t^{switch}}\right) < -50\right) \end{cases} \tag{7.8}$$

where $t^{switch} > 1$ is a pre-defined change step, and $x_{t^{switch}}$ is a single solution produced at t^{switch} by the solver. Equation 7.8 means that with $t \leqslant t^{switch}$, the form of $\widehat{F}(x_t)$ is always equal to $f^1(x_t)$; with $t \geqslant t^{switch}$, depending on the solution of $x_{t^{switch}}$ the form of $\widehat{F}(x_t)$ would switch to either $f^2(x_t), f^3(x_t)$ or $f^4(x_t)$. In other words, the Equation 7.8 defines the switching rule of the problem.

Because the sub-function f^1 of Equation 7.6 is a time-linkage problem and because at any $t \geqslant t^{switch}$ the the values of the function $\widehat{F}$ depend on the solution found by the solver at t^{switch}, the main problem in Equation 7.6 is also time-linkage. Equation 7.6 is also a prediction-deceptive problem because it will deceive any good predictor to choose a high-value trajectory during the period $\left[0, t^{switch}\right]$. After the change step t^{switch}, however, such a high-value trajectory

[3] The concepts like *full-description forms*, *time-dependent parameters* and *dynamic drivers* have been defined and described in Chapter 4.

may lead the solver to the worst possible trajectory of $f^2(x_t, t) = -60$, which may eventually affect the total score of the solver and make a solver with predictor to perform worse than a solver without a predictor!

It should be noted that for the purpose of simplicity in this test problem I include only one function-form switch and the switching time is assumed to be fixed (as found in the real-world applications in (Sonntag *et al.* 2008, Summers & Bewley 2007)). In reality, there might be more than one switch and the switching time might not be fixed but to be determined by the solver or automatically by the dynamic behaviour of the system. However, the prediction-deceptive property in these scenarios should still be the same as in the simple test problem we are considering.

The problem is named DTP2. Experiments and discussions on this prediction-deceptive problem will be presented in Subsection 7.4.3.

7.4.2 Test algorithms

To carry out the experiments, I developed three different versions of GA to represent the three different approaches in solving time-linkage problems: first, a standard GA (Algorithm 14, page 235) to represent the tradition *optimise-the-present* approach; second, a combination of GA + predictor (linear least-square regression) to represent the predict-the-future-based-on-history-data approach proposed in (Bosman 2007) (Algorithm 15, page 235); and third, a combination of GA + predictor + knowledge (about the switching rules) to represent the approach proposed in Section 7.3 (Algorithm 16, page 236). It should be noted that, for the sake of simplicity Algorithm 16 was designed to solve only the cases where the switching rules are known and the switching time is also known. For GA+Predictor and GA+Predictor+Knowledge, I chose the least-square fitting technique as the predictor method. The assumption for the method is that the function to be estimated has a quadratic form. Of course in reality this assumption is not always true and it might be necessary to estimate the form of the function as well. In such case, powerful function approximation models like neural networks can be used to represent the function to be predicted. Here I use the simple assumption that the function form is quadratic because our purpose is not to propose a state-of-the-art or an efficient algorithm but just to show aproof of principle that EAs with predictor can achieve better results than EAs without predictor in normal DTPs, that EAs with predictor might be deceived in prediction-deceptive

DTPs, and that by taking into account the future switching rules when predicting we can help EAs to avoid being deceived.

Algorithm 14 Standard GA

1. *Initialisation*
2. *Search*: for each generation
   (a) Standard GA's crossover
   (b) Standard GA's mutation
   (c) Evaluation: For each individual $x(t_{now})$, evaluate $f(x(t_{now}))$
   (d) Standard GA's selection

Algorithm 15 GA + Predictor

List of parameters:

Pred:	A linear least-square regression to approximate quadratic functions
s	Change severity
h^{len}	The length of the predicted future horizon

1. *Initialisation*
2. *Prediction*: After m generations, use the predictor *Pred* to estimate the current function form based on history data
   - Input:
     (a) Solutions achieved in previous $1/s$ change steps: $\forall x(t), (t^{now} - \lfloor 1/s \rfloor) \leqslant t \leqslant t^{now}$.
     (b) All corresponding function values $f(x(t))$ and the corresponding change step t.
   - Output: the estimated function form f_{pred}
3. *Search*: for each generation
   (a) Standard GA's crossover
   (b) Standard GA's mutation
   (c) Evaluation: For each individual $x(t_{now})$, evaluate
   $$\text{Fitness}(x(t_{now})) = \left\{ f(x(t^{now})) + \sum_{t=t^{now}+1}^{t^{now}+h^{len}} f_{pred}(x(t)) \right\}$$
   (d) Standard GA's selection

To create a fair testing environment, all three algorithms use the same set of parameters. Table 7.1 shows the detailed parameters of the algorithms and all other settings for the experiment.

To evaluate the performance of the algorithms, I use two measures. The first one is *perfor-*

Algorithm 16 GA + Predictor + Knowledge about the switching rules

List of parameters:

Pred:	A linear least-square regression to approximate quadratic functions
s	Change severity
h^{len}	The length of the predicted future horizon
f_{switch}	Expected full-description form of the switching rules
$\{T_1, ..., T_{n-1}, T_n\}$	The set of switching times within the current horizon $t^{now} < T_i \leqslant t^{now} + h^{len}$

1. *Initialisation*
2. *Prediction*: Same as step 2 in Algorithm 15.
3. *Search*: for each generation
   (a) Standard GA's crossover
   (b) Standard GA's mutation
   (c) Evaluation: For each individual $x(t_{now})$,
      i. *Calculate current function value*: $A = f(x(t^{now}))$
      ii. *Calculate the expected future function/variable values until the first switching time*:
      $B = \sum_{t=t^{now}+1}^{T_1} f_{pred}(x(t))$
      iii. Estimate the variable $x(T_1)$ given the estimated outcome of f_{pred} during the period $[t^{now}+1, T_1]$
      iv. *Calculate the expected future values after the first switching time*:
      $C = \sum_{T_i=T_1}^{T_{(n-1)}} \sum_{t=T_i+1}^{T_{(i+1)}} f_{switch}(x(t), x(T_i))$
      v. *Calculate the fitness value of* $x(t_{now})$: $\text{Fitness}(x(t_{now})) = A + B + C$
      vi. *Update*: update the set of switching times for the next future horizon
   (d) Standard GA's selection

Table 7.1: Test settings for GA, GA+Predictor and GA+Predictor+Knowledge.

Algorithm parameters	Pop size	25
	Elitism	No
	Selection method	Non-linear ranking
	Mutation method	Uniform, $P = 0.15$
	Crossover method	Arithmetic, $P = 0.8$
	Prediction method	Least-square regression for quadratic function
Test problem settings	Number of runs	50
	Change frequency	25 function evaluations (one generation)
	Change severity s	0.001
	Learning frequency	Every 10 generations
	Number of change steps	$11/s$ ($11,000$ change steps, $t^{end} = 11,000$)
	Length of predicted future horizon h^{len}	$5/s$
	Switching time	$8/s$

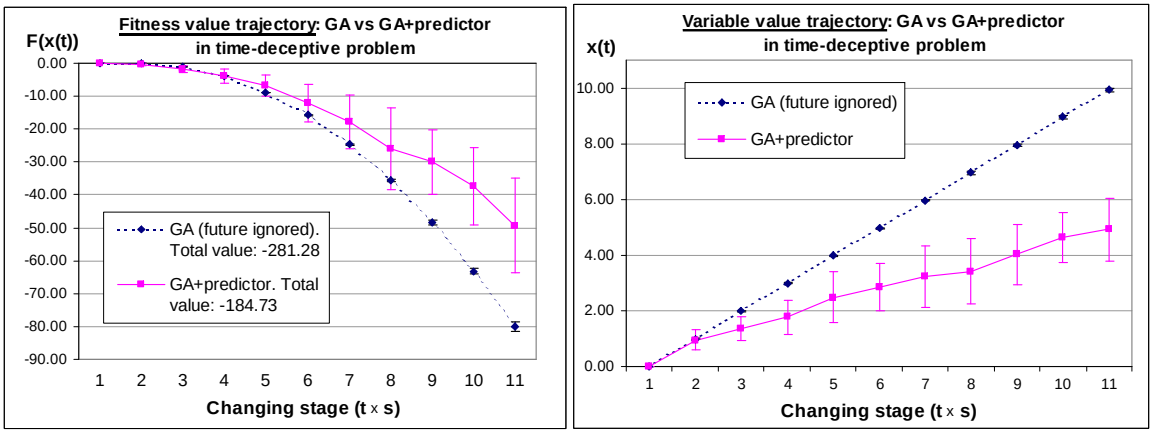

Figure 7.3: Plots of the mean (and standard deviation) of highest function values over 50 runs: GA without predictor vs GA with predictor in a time-deceptive problem (DTP1)

mance plot - the plot of the trajectory of the best function values that the algorithms achieved at each change step. I also plotted the trajectory of the variable $\mathbf{x}$ as time goes by to study the behaviours of the algorithms. The second measure is the total function values, which is calculated as the summation of the best function values taken after each $1/s$ change steps ($1,000$ change steps): totalVal $= \sum_{i=1}^{10} f\left(x\left(t^{begin} + \lfloor i/s \rfloor\right)\right)$. Detailed experimental results are given in the next subsection.

7.4.3 Experimental results

GA vs GA+Predictor in time-deceptive problems (DTP1)

Here I verify the suggestion of Bosman (2007) that in time-deceptive DTPs, learning from the past to predict the future can be useful. Figure 7.3a, where the mean and standard deviation of function values of GA and GA+Predictor in the problem DTP1 are shown, confirms the advantage of this approach. The figure shows that although GA+Predictor has worse function values in the first few change stages, in the longer run it perform much better (has higher total values) than the traditional GA, which only focuses on optimising the present. The results confirm the advantage of maximising future values over just optimising the present in this particular problem.

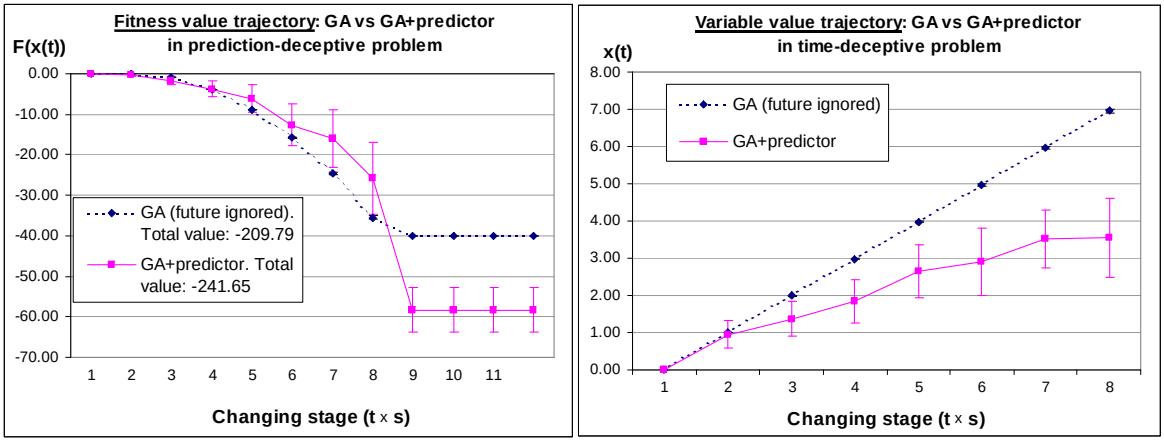

Figure 7.4: Plots of the mean (and standard deviation) of highest function values over 50 runs: GA without predictor vs GA with predictor in the prediction-deceptive problem (DTP2)

GA vs GA+Predictor in prediction-deceptive problems (DTP2)

Predicting the future using data from the past, however, is not always beneficial in solving DTPs. In problems like the DTP2 where a high function value might switch the system to a low-value trajectory and vice versa, predicting future using data from the past might make the algorithm perform worse than not using a predictor. This behaviour is confirmed in the experiment. Figure 7.4a shows that GA+Predictor actually has lower total values than GA. This is due to that, since the eighth changing stage, the high-value trajectory that GA+Predictor predicted during the period $[0, t^{switch}]$ leads the algorithm to a worse trajectory than what GA achieves.

GA vs GA+Predictor vs GA+Predictor+Knowledge in prediction-deceptive problems (DTP2)

Here I verify the efficiency of our proposed approach described in Section 7.3, which suggests that the knowledge of the switching rules, if available, should be taken into account when anticipating the future. Figure 7.5a shows that the new approach does help improve the performance of the algorithm (GA+Predictor+Knowledge) and avoid being deceived into the wrong trajectories. As can be seen in Figure 7.5a, during the first six changing stages GA+Predictor+Knowledge follows exact the same trajectory as GA+Predictor to maximise the function value trajectory in the period when the system has not switched to the other mode yet. However, from the sixth changing stage, GA+Predictor+Knowledge follows a different route from the original

GA+Predictor. At the sixth changing stage, GA+Predictor+Knowledge chose a slightly higher function value, which leads it to a completely different route from GA+Predictor and normal GA at the seventh changing stage. At this stage, the algorithms chose a very low function value, which is achieved thanks to the high value it chose in the previous changing stage. Although GA+Predictor+Knowledge has to sacrifice its current performance to achieve such a low function value, this low value helps the algorithm to reach to a better trajectory after the switch and eventually it has a significantly higher total function values than GA and GA+Predictor. This good result confirms the usefulness of anticipating future function-form switches when solving DTPs. The behaviour of GA+Predictor+Knowledge in choosing the variables to achieve a high total function value is also shown in Figure 7.5b.

Another note is that when taking into account the future, the problem becomes more complex toward GAs and it is getting more difficult to get high precision results, as can be seen by looking at the standard deviations of the results in Figures 7.3, 7.4, 7.5 . We can see that the traditional GA (future ignored) achieves very consistent results (standard deviations of the mean best values are almost zero) over 50 runs. However, when the algorithm has to predict the current function-form (GA+Predictor) and hence has to optimise not only the present but also the future, the problem becomes more complex and the standard deviations of the mean best values over 50 runs becomes higher. When the algorithm has to predict the current function-form *and* also has to anticipate any possible future mode-switching, the problem becomes even more complex and hence the level of inconsistency (standard deviation) increases even higher. This phenomenon shows the trade-off in taking into account the future when solving DTPs.

7.5 Summary

Although it was believed that time-linkage problems can be solved to optimality by relying on history data of the algorithm to predict the trajectory of future function values, in this chapter I pointed out a challenging class of time-linkage problems where this prediction approach might fail to find the optimal results. We named this class *prediction-deceptive time-linkage problems*.

I also suggested an approach to solve this class of problem under certain circumstances and developed algorithms to implement this approach. Experiments were also made to verify the advantage and disadvantage of the anticipation approach in solving DTPs, to illustrate the impact of the prediction-deception property on algorithm performance, and to evaluate the

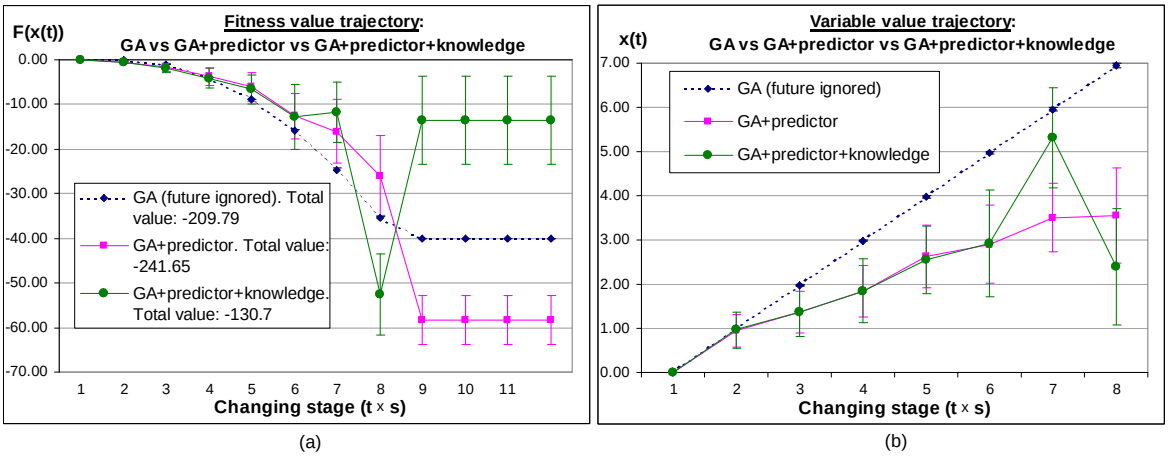

Figure 7.5: Plots of the mean (and standard deviation) of highest function values over 50 runs: GA without predictor vs GA+predictor vs GA+predictor+switching_knowledge in the prediction-deceptive problem DTP2

efficiency of our proposed approach in solving prediction-deceptive time-linkage problems.

Two time-linkage benchmark problems were also proposed in this chapter (Section 7.2 and Section 7.4.1). The benchmark problems are able to simulate the known (time-deceptive) and unknown (prediction-deceptive) properties of time-linkage problems and can be configurable, making it easier for researchers to test their existing algorithms.

Although the experiments (and the algorithms + test problems) in this chapter are over-simplified, and the advantages of a predictor/ predictor+knowledge are expected, such simplifications are necessary to proof the principle and to show the potentiality of EAs because this research is just a beginning step and is the first EDO study in this topic. To the best of our knowledge, previously this class of problems has not been taken into account in existing academic EDO research despite their popularity in real-world scenarios (as shown in Chapter 3).

Chapter 8

Conclusions and future work

The thesis has at least partially provided answers for the questions raised in Section 1.3 about what types of DOPs have been covered by existing EDO academic research and if there are any missing links between academic EDO research and real-world applications. Based on these answers this thesis has focused on studying some important issues to help close some of the existing gaps in EDO academic research. These issues are defining DOPs and solving continuous DCOPs and DTPs - two classes of problems commonly found in real-world scenarios but have rarely been studied in EDO. The results of this thesis in continuous DCOPs and DTPs, which are among the first in these areas, provide a deeper understanding of the unknown characteristics and the solvability of these problems, and suggest some promising ways to solve these challenging problems using EDO techniques.

8.1 Summary of Major Contributions

Detailed of the contributions in this thesis have been described at the end of each chapter. Here the most significant contributions are summarised as follow:

1. Identify for the first time the important gaps between real-world DOPs and EDO academic research, including the current coverage of EDO academic research, the types of problems that have not been covered by the EC community, the characteristics and problem information that we can used to solve DOPs more effectively, and the way that DOPs are solved in real-world scenarios.

2. Provide a new definition framework, new sets of benchmark problems (for DCOPs and

DTPs) and new sets of performance measures (for DCOPs and DTPs) to better characterise the unknown factors of DOPs.

3. Develop novel approaches to solve continuous DCOPs, an important and common class of DOPs but have not been studied in EDO research. The new approaches are developed based on detailed analyses (including some counter-intuitive findings) on the representative characteristics of DCOPs, the strengths/weaknesses of existing EDO/CH methods in solving DCOPs, the influence of different algorithmic components on algorithm performance, and on my proposed list of requirements that an algorithm should meet to solve DCOPs effectively.

4. Develop a new approach to solve DTPs, another important and common class of DOPs but have not been well-studied in EDO research. The approach is also developed based on analyses on the characteristics of DTPs and the strengths and weaknesses of existing EDO methods in solving DTPs.

8.2 Future Work

There are many related research topics that can be pursued in the future to improve and further evaluate the results of this thesis. The survey of real-world applications in Chapter 3 shows that there are many open research areas to bring academic EDO research closer to real-world applications. Among these areas, some possible interesting future research directions are:

1. Focusing more on some particular types of problems such as DCOPs and DTPs, which are common in real-world scenarios but have not attracted enough attention from the EC community yet;

2. Re-defining the optimisation goals, performance measures and benchmark problems in academic EDO research to better reflect real-world situations;

3. Studying those characteristics of real-world problems that have not received much interest from the EC community. Examples of such characteristics are changes in constraints, changes in number of variables, the predictability and detectability (or the lack thereof) of changes.

4. Studying the efficiency and suitability of EAs in different types of real-world applications;

As to the particular research topics that I have studied in this thesis, because the works that I have done in the thesis are only among the first steps in these topics, there are a lot of future works to be addressed. Some possible directions for future extensions are discussed below:

1. *The definition framework*: In this thesis the definition framework has mainly been used to answer the question of how to characterise DOPs (i.e. how to distinguish DOPs, to encapsulate the dynamic behaviours and the changing factors in the definition and to separate the static aspects from the dynamic aspects) and how to generate dynamic benchmark problems. Because the definition framework facilitates the inclusion of optimisation algorithms and the separation of dynamic components from the static ones, one of possible future works might be to use the framework to study the difficulty of each dynamic/static component and how the difficulty of each component affect the overall difficulty of the problem toward a specific algorithm. Because the framework defines each aspect of a DOP to a more detailed level, another future direction is to use the framework as a basis for theoretical research in dynamic optimisation.

2. *Benchmark problems*: In this thesis a set of 18 benchmark problems for DCOPs (G24) and two benchmark problems for DTPs have been proposed. Although these benchmark problems are effective in analysing the behaviours of the tested algorithms, they are mostly based on unimodal static problem instances and this might limit their generality. A possible future direction is to integrate the existing benchmark problems in this thesis with the multimodal, scalable benchmark problems that I have developed for DCOPs (Nguyen 2008*a*) and DTPs (Nguyen 2008*b*, pp. 26-29).

3. *Analysis of the performance of existing methods in DCOPs*: The analyses in Chapter 5 and Chapter 6 only considered some basic and representative DO and CH strategies. Future extension of these analyses should consider a broader range of strategies, including the memory-based approaches and the current state-of-the-art methods in DO and CH. In addition, I plan on extending the dynamic settings of the analyses to test the algorithms in different types of changes and different values of change frequency, change severity, population size, and evolutionary parameters. Analyses on these wider ranges of parameters will be included in the revised version of (Nguyen & Yao 2010*a*). It would also be interesting to systematically study the situations where the presence of constraints and dynamics would

make it easier for certain classes of algorithms to solve DCOPs.

4. *New methods to solve DCOPs*: One of the future directions is to test the algorithms proposed in Chapter 6 on the multi-modal benchmark set in (Nguyen 2008*a*). I plan to carry out more detailed analyses on the performance of the algorithms under a wider range of test settings with different change frequency, change severity, population size, and evolutionary parameter values. I am also interested in hybridising the proposed algorithms with state-of-the-art constraint handling methods such as stochastic ranking (Runarsson & Yao 2000).

5. *Dynamic time-linkage optimisation*: The work in this thesis is just an initial step in an attempt to understand more about DTPs and to solve this class of problems effectively. For future works we plan to do more experiments on more realistic scenarios with a more powerful predictor integrated with state-of-the-art EAs. Especially, more research will be carried out to investigate the situation where the algorithm needs to determine multiple switching times during the optimisation process. The possibility of combining time-linkage handling techniques with normal environmental dynamic handling techniques will also be investigated. A further goal will be to carry out experiments on real-world problems like hybrid systems. An investigation on the relationship between the time-linkage property and co-evolution will also be carried out in the future.

Appendices

Table 1: Combinatorial real-world references that use EA/metaheuristic methods

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / meta-heuristics?	Predictable	Visible	Constr. prob-lems?	Single/ Multi-obj	Optimisation goal	Types of dynamics	Restart/ Track	Factors that change: Parameters of obj func	Factors that change: Domain range	Factors that change: Number of vari-ables	Factors that change: Other pa-ra-me-ters	Constraints
Take-off runway schedul-ing (Atkin *et al.* 2008)	Real data of London Heathrow air-port, provided by National Air Traffic Services	The problem here is to find the optimal take-off runway schedul-ing, taking into account not only aircrafts in the holding area but also the taxiing ones. Although real-world data is used in this problem, the data is used as a static benchmark in which the author tests several dy-namic scenarios rather than to in-vestigate the real dynamics. In this reference we only consider those characteristics that evidently ex-ists in real-world situations	.Yes. ((1) The current schedule of aircrafts might af-fect how the future solution would be (2) The movement of one aircraft might block the way of another (3) Changing the take-off aircraft position would be harder if the position is closer to the take-off time. The first type of time-linkage is not taken into account (UNHANDLE). The second is dealt with using some heuristic rules and the third property is dealt with by fixing the aircraft position to be at least two minutes before take-off time (HANDLED)	..Partly... (solved by Tabu search and heuris-tics)	Partly. (some factors like taxiing time can be pre-dicted)	Partly:.No. (.INVIS-IBLE. - uncer-tainty, or changes, occur due to the mis-prediction of aircraft's taxiing time) &..Yes.. (.VISI-BLE. - the system is noticed whenever a new plane arrives or if more in-formation becomes available (and hence less uncer-tainty))	..Yes.. (mostly soft con-straints)	.S. (mul-tiple objectives but they are pri-oritised by weight and then combined together to create a single objective. The rea-son is to provide users with only one single so-lution to simplify users' tasks)	Optimality & Previous-solution dis-placement restriction (deviation from previous solution will be penalised) & Quick re-covery & Spec Satisfaction (changing the take-off aircraft posi-tion becomes harder when it is the closer to take-off time, so they fix the aircraft poistion to be at least two minutes before take-off time) are all men-tioned but it is not clear about the priority	.N/I. (the simulated data fol-lows a linear dis-tribution, but it is not clear if this would happen in real-world situa-tions)	.Tracking. (.DISP.: new solutions are based on previous ones to avoid dis-turbances.)	..Yes.. (new aircraft might arrive and hence change the ex-isting take-off order)	.N/I.	..Yes.. (no detail is given but the number of vari-ables (corre-sponding to the number of air-crafts being scheduled at one time) should be variable depend-ing on the ar-rival and taking off of aircrafts)	.N/I.	..Yes.. (the soft con-straints depend on the val-ues of the predicted take-off time, take-off timeslot, etc of each aircraft. These val-ues would change when a new aircraft arrives or when an existing air-craft leaves. In addition, some of the constraint parame-ters are uncertainty due to the uncertain of taxiing time)
Ship Schedul-ing (Mertens *et al.* 2006)	Real static data of 7 transport ships from Tractebel Engineering company	The problem here is to find the op-timal schedule for ships. Although real-world data is used in this problem, the data is used as a static benchmark in which the author tests several dy-namic scenarios rather than to in-vestigate the real dynamics. In this problem we only consider those characteristics that evidently exists in real-world situations. Another note is that the use of tracking/memory (DynAWC) might not work well in the case of dy-namic constraint	.N/I.	.Yes. (ACO)	.N/I.	.Yes. (.VISI-BLE.)	.Yes. (69 vari-ables and 101 con-straints, of which 75 are hard)	.S. (no detail of objective function is given)	Optimality ; Previous-solution dis-placement restriction (it is unclear about their priority.)	.N/I.	.Tracking. (hybridised with ran-domising (the best results was achieved when pheromone quantities are set to the average of the previous pheromone quantity and the default quantity. Tracking was used to keep the new solution close to the old one -.DISP.)	.N/I. (ships might be de-layed due to storms or other unex-pected events. However it is not clear how does it affect the obj func-tion)	.N/I.	.N/I.	.N/I.	.Yes. (ac-cording to the pa-per the storm event changes the constraints)

Table 1 Combinatorial EA/metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / meta-heuristics?	Predictable	Visible	Constr. prob-lems?	Single/ Multi-obj	Optimisation goal	Types of dynamics	Restart/ Track	Factors that change				
												Parameters of obj func	Domain range	Number of vari-ables	Other pa-ra-me-ters	Constraints
Document Stream Mod-elling problem (Araujo & Merelo 2007)	Document streams with keyword "gmail" from www.blogalia.com (the comments sent to all blogs hosted in Blogalia from Jan 2002 - Jan 2006)	The problem here is to search for the most suitable model to repre-sent the current trend in docu-ment streams. It is mentioned that restart is the approach that provide the most accurate result, but not satisfiable in real-time situations. This shows that tracking optima is chosen over restarting not only because it might produce better results (which might not always be the case) but also because it can produce good result in shorter time	.N/I. (Not men-tioned)	.Yes. (GA). The mem-ory ap-proach is also ap-plied be-cause there are re-cur-rent changes	.Partly. (it might not be possi-ble to predict 'when' and 'how' the changes hap-pen, but once a change hap-pens, it might be pos-sible to approx-imate the func-tion form)	.No. (.IN-VISIBLE. - the op-timisation process is divided into time windows. Each time window represents a segment of time for which enough data are available). The algo-rithm still needs to detect if a change happens in each time window though.	.Yes., soft con-straint (con-straint is repre-sented as the penalty cost of transit from one state to another amd the penalty cost is in-cluded in the obj func-tion)	.S.	(1) quick re-covery (2) Op-timality.	The ar-rival of document stream is believed to follow expo-nential distri-bution. There are also other changing rules. Changes might be recurrent.	.Tracking. to provide a de-cent solution quickly using memory el-ement from previous run (.QUICK.). The previous knowledge is also used to predict the trend of data (.LEARN.)	.Yes. (changes $\sim$ drifts in the concept (topic of docu-ments))	.No.	.Yes. (the individ-uals are variable-length sequence)	.N/I.	.No.
Evolvable hard-ware problem (Tawdross *et al.* 2006)	An opera-tional am-plifier in an environ-ment with changing heat (lab-control)	The problem is to evolve the hardware struc-ture to make it as close to the specification as possible. When a change happens, the hardware structure needs to be evolved a gain to match the specification. The authors also mentioned other possible dynamic factors in real-world situations as manufacture tolerance or aging of devices	.N/I. (Not men-tioned)	.Yes. (PSO)	.N/I.	.No. (.IN-VISIBLE.)	.Yes. (the specifi-cation are con-straints while the error func-tion is the penalty func-tion)	.S.	(1) Previous-solution dis-placement restriction (more impor-tant because the new hard-ware structure need to be similar to the starting one to comply with indus-trial standard. The authors mention that solutions achieved using the restart approach have arbitrary structures which might be difficult to be accepted by the indus-try) and (2) Optimality ;	One type of changes is noises. There are also other problem-specific changing rules: (tem-perature changes lead to changes in the state of the chip)	.Tracking. to make sure that the dis-palcement property is maintained (.DISP.)	.Yes. (tem-perature changes lead to changes in the state of the chip)	.No.	.No.	.N/I.	.No. (the chip spec-ification should be fixed re-gardless of changes in the environ-ment)

Table 1 Combinatorial EA/metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / meta-heuristics?	Predictable	Visible	Constr. problems?	Single/ Multi-obj	Optimisation goal	Types of dynamics	Restart/ Track	Factors that change: Parameters of obj func	Factors that change: Domain range	Factors that change: Number of variables	Factors that change: Other parameters	Constraints
Dynamic route planning for car navigation system (Kanoh 2007, Kanoh & Hara 2008)	Real-world map (Tsukuba city and Northen Tokyo) from Navigation System Researchers' Association (www.naviken.jp) and the movement of 28000 cars from Nagel & Rasmussen's model (this model is not real-world, only simulates some behaviours from certain, rather simple situations).	The problem here is to find the optimal route (with least travel time) to a given destination for cars. Some of the test data are artificially generated without evidence of whether they are based on real-world data or not. For this reference we only consider the data that evidently exist in real-world situations	.Yes. (UNHANDLE - in the context that the routing decision might lead to a new network with more/less congestion. This situation was briefly mentioned in this reference when the disadvantage of the Dijkstra algorithm was discussed (page 70). However, it is not fully considered in the paper except an effort made to avoid congestion by not applying the same optimal decision to all vehicles in the road)	.Yes. (Virus GA)	.N/I.	.Yes. (.VISIBLE.)	.Yes. (there are both hard (traffic rules) and soft constraints (prefer wide, large road, reduce number of turns, signals etc).	.S.	Quick recovery & Optimality. No priority is clearly given but for two algorithms (AIS&DA) with the same optimal result, the one with faster recovery (AIS) is chosen	Changes follow specific rules such as vehicle speed, road information etc. Changes might be recurrent. That is why some part of the route can be reused for the next change by using AIS	.Tracking. (.DISP. - because the new solution (new route) need to be based on a part of the existing solution (current route)..QUICK. - Also tracking helps to meet the requirement of quick recovery)..LEARN. - knowledge from the past (some part of the routes) is also used to learn/anticipate changes in the future	.Yes. (vehicle speed, road information)	.N/I.	.Yes. (the individuals are variable-length sequence)?	.N/I.	.No.
Survival routing in dynamic DWM network (Ngo *et al.* 2006)	Real-world NSFNet topology; artificial distributions for changes. However those distributions are drawn from real-world observations.	The problem here is to find the optimal route between a pair of nodes in the DWM network	.Yes. (UNHANDLE - in the context that the routing decision might alter the existing free nodes/links of the network. This situation however is not considered by the authors.)	.Yes. (ACO)	.N/I.	Partly:.Yes. (arrival connection requests are visible (.VISIBLE.) &.No.: network congestions are not visible (.INVISIBLE.) and need to be detected by having the routing table updated continuously by mobile agents)	.Yes. (not all links are feasible)	.S.	Optimality (routes with minimal blocking probability are preferred) and Quick recovery (the algorithm needs to maintain small setup delay). No clear indication of the priority for each goal, but optimality is the criteria to asses algorithms in the experiment.	Changes in connection request arrival follows Poisson distribution; Changes in session holding time follows exponential distribution; Changes in place of request follows uniform distribution.	.Tracking. (because the new solution (new route) need to be based on a part of the existing solution (current route) -.DISP.. Also tracking helps to meet the requirement of quick recovery)- .QUICK.	.Yes. (new arrival connection or existing ones disconnected; congestion; link failures; etc)	.N/I.	.N/I.	.N/I.	.Yes. (some feasible nodes might become infeasible and vice versa over time)

Table 1 Combinatorial EA/metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / meta-heuristics?	Predictable	Visible	Constr. prob-lems?	Single/ Multi-obj	Optimisation goal	Types of dynamics	Restart/ Track	Factors that change				
												Parameters of obj func	Domain range	Number of vari-ables	Other pa-ra-me-ters	Constraints
Airline Schedule Recover Problem (recover from airport tem-porary closure) (Liu *et al.* 2007)	Flight sched-ule of a Taiwanese do-mestic MD90 fleet in one day when one airport is temporarily closed in one hour.	The problem here is to find the opti-mal schedules for the airline after a temporary clo-sure of airports. It is not clear if the change is gen-erated artificially or if it is re-ally a real-world event from exist-ing data	.N/I.	.Yes. (EPGA)	.N/I.	.Yes. (.VISI-BLE.)	.Yes.. (There are 6 con-straints de-scribed in page 2400. Some other con-straints are repre-sented as con-current objec-tives in the prob-lem formu-lation)	.M.	(1) Previous-solution dis-placement restriction and Optimality. Both criteria are included as objectives and are used as measures in evaluating the algorithm. (2) Quick recorvery: this goal is cited as one of the goals in disruptive management. It is also used as one of the criteria to evaluate the algorithm.	.N/I.	.Tracking. (because the new solution needs to be based on a part of the exist-ing solution -.DISP.)	.Yes. (one airport closes tem-porarily for one hour). This would directly affect the "flight connec-tion" objec-tive and the "duty swap" objec-tive	.No.	.Yes. (the chro-mosome length is deter-mined by the number of air-craft and airport simulta-neously. If so in case number of airport changes, dimen-sion should change accord-ingly)	.No.	.Yes. (Con-straints 3-6 in page 2400 are dynamic. In addition if we consider the "flight connection" and "duty swap" ob-jectives as constraints, they are dynamic too.)
Dimen-sioning and load balanc-ing for multi-topology Interior Gateway Protocol traffic (Wang, Ho & Pavlou 2008)	The proposed method was tested using real-world topology and traffic data from GEANT and Abilene networks	The optimisa-tion include two phases: the off-fline phase is carried out on a weekly or montly timescale to set up the network link weights to maximise the intra-domain path diversity across multiple routing topolo-gies, and the online phase to dynamically balance the load based on the pre-defined link weights to deal with changes	.Yes. , partly (HANDLED - it is showed that the decision of changing link weights influences future potential of congestion. This time-linkage effect is handled offlineby choosing the optimal link weights to avoid future congestions. The dynamic bal-ance mechanism (main topic of the paper) also influence the fu-ture problem by controlling and avoiding future congestions.)	.Offline. only (GA is used for the off-fline phase to find the opti-mal link weights. The on-line phase is solved us-ing a special-purpose heuris-tic)	.Partly. (con-gestion can be partly pre-dicted and con-trolled by choos-ing the optimal link weight and dynam-ically balance the traffic load)	.Yes. (.WIN-DOW. - time-window approach)	.Yes. (there are con-straints for links, nodes, topolo-gies and band-with capac-ity)	.S.	Optimality (minimising congestion) Previous-solution dis-placement restriction (the aim of the method is to avoid frequent and on-demand reassignment of link weights, hence min-imising large difference in the solution before and after each change)	The traf-fic data shows that traffic peaks are periodical	.Tracking. (not all traf-fic flows are reset but only the most utilised link is chosen and the load in that link will be lighten gradually by shifting some traffic flows in that link to other alternative less-utilised paths. The purpose of tracking is to avoid sud-den changes to existing connections -.DISP..)	.Yes.	.N/I.	.N/I.	.N/I.	.N/I.

Table 1 Combinatorial EA/metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / meta-heuristics?	Predictable	Visible	Constr. prob-lems?	Single/ Multi-obj	Optimisation goal	Types of dynamics	Restart/ Track	Factors that change				
												Parameters of obj func	Domain range	Number of vari-ables	Other pa-ra-me-ters	Constraints
Aircraft landing problem (Moser & Hendtlass 2007*b*)	A set of air-craft landing benchmark problems from (Beasley *et al.* 2004). Although the problems are artificial benchmark, it seems that they have the characteris-tics widely believed to belong to real-world scenarios. For example, the arrivals of air-crafts follow a negative exponetial distribution; the speed of aircraft is similar to real-world ones, separation dis-tances/times on land-ing were calculated from real-world data of Heathrow airport, etc.	The displace-ment cost is incorporated into the objective function. In this problem quick recovery is possibly the most preferred optimisation goal because the pa-per shows some examples where more powerful algorithms are discarded due to their disadvan-tages in meeting certain time limit).	.Yes. (Decision of which aircraft to land first would change the problem in the next time step, but that property is not taken into account when solving the problem - UNHAN-DLE. In addition, there is a "displace-ment resitriction" (see Beasley *et al.* (2004))" of how the next solution should be given the current solution so that the next solution is not too different from the current one - HANDLED)	.Partly. (solved by Ex-tremal Op-ti-miza-tion and other heuris-tics)	.No.	.Yes. (.WIN-DOW. - the whole optimi-sation process is divided into smaller time win-dows. The problem is considered changed after each time win-dow)	.Yes.	.S.	1: Quick re-covery (to meet the limit of the time window); 2: Previous-solution dis-placement restriction & Optimality (these two costs are all integrated in the obj func); 3. Spec Sat-isfaction: all planes need to land within a specific time-frame.	The ar-rival of aircrafts follows a negative expo-nential distri-bution. There are also other problem-specific changing rules.	.Tracking. intial solu-tion at each time window is created by adjusting the previous found solu-tion, plus any newly appearing airplane. No clear reason why tracking is chosen, but as stated in the used benchmark, previous-solution displacement restriction is a requirement for problems of this type -.DISP.)	.Yes.	.No.	.Yes.	.No.	.No.
Hydro-thermal Schedul-ing Problem (Deb *et al.* 2007)	The static part is from a problem claimed by the authors as a "real-world" problem. It is described in (Basu 2005) and originates from a PhD thesis that we do not have access. However, the dynamic (changes in the demand) is simulated artificially	The problem here is to find the optimal alloca-tion of power to electricity generators to minimise the fuel cost of thermal generation and emission prop-erties. For this reference we only consider those characteristics that evidently ex-ists in real-world situations	.N/I. (Not men-tioned in the paper.)	.Yes., RI and Hy-perM with NSGA-II	.N/I. (data are gener-ated artifi-cially)	.Yes. (.WIN-DOW. Pseudo vis-ible - the whole op-timisation process is divided into smaller time win-dow. The problem is considered changed after each time win-dow)	.Yes.	.M. (cost of thermal genera-tion and thermal emission)	Optimality ; Quick recov-ery (because it is men-tioned that the optimal so-lution should be tracked as quick as possible)	.N/I. (although changes in the tested problem is re-current, data are generated artifi-cially. This type of change is likely realistic though)	.Tracking. (.QUICK. - to get a good solution quickly)	None	None	None	None	.Yes. (power demand)

Table 1 Combinatorial EA/metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / meta-heuristics?	Predictable	Visible	Constr. prob-lems?	Single/ Multi-obj	Optimisation goal	Types of dynamics	Restart/ Track	Factors that change				
												Parameters of obj func	Domain range	Number of vari-ables	Other pa-ra-me-ters	Constraints
Optimising supply-chain configu-rations (Akanle & Zhang 2008)	The consid-ered problem was adapted from real-world supply-chain data of a Fortune 100 corporation, described in (Graves & Willems 2005)	This approach does not try to deal with each single order indi-vidually (provide optimal supply chain for each order) but to provide a stable supply chain for a number of or-ders in a certain period of time in the near future. It does that by looking at the optimum supply chain for each of a set of sequence orders, then try to extract their common properties and design a main supply chain for these common properties)	.Yes. (HANDLED)	.Partly. (GA to tune the op-era-tional pa-ra-me-ters for each indi-vid-ual or-der)	.Partly. (It is as-sumed that future de-mand could be mod-elled; and varia-tions in costs and lead-times of re-soruces could be pre-dicted)	.Yes.? (.IN-VISIBLE. - change must be detected. However, it is de-tected by other com-ponents (agents), not by the opti-miser itself. In other words, to-ward the optimiser changes are visible)	.Yes.	.M. (the objec-tive is to find the minimum number of tardy or-ders at the minimum cost)	Optimality ; Quick recovery (it is required that the or-der need to be complete before a given deadline re-quired by customers); Spec Satisfac-tion (in the sense that the solver aim at provide supply-chain structure to keep up with expected re-quirements and capacity in the future)	.N/I.	.Tracking. (in the sense that previous knowledge is used to learn the optimal chain struc-ture for the near future -.LEARN.)	.Yes. (it is men-tioned that the num-ber of feasible resource option as well as the cost of each feasible resource option may change over time. How-ever, most of these changes are not con-sidered in the current experi-ment)	.N/I.	.Yes. ((it is men-tioned that the num-ber of feasible options of each node, and the number of nodes may change over time. However, most of these changes are not con-sidered in the current experi-ment)	.N/I.	.Yes. ((it is mentioned that the capacity and feasibil-ity of each resource node may change over time. The processing time for each type of order may also change. However, most of these changes are not con-sidered in the current experiment)

Table 2: Continuous real-world references that use EA/metaheuristics

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / metaheuristics?	Predictable	Visible	Constr. prob- lems?	S/M-obj	Optimisation goal	Types of dynamics	Restart/ Track	Factors that change: Parameters of obj func	Factors that change: Domain range	Factors that change: Number of variables	Factors that change: Other prm	Constraints
Adaptive Farming Strategies (Jin *et al.* 2007)	UK Farming Data and UK agriculture subsidy policy are used in the simulation to forecast future land-use decisions.	The problem here is to maximise income by appropriately choosing mixed grazing strategy. The pattern of change is observed from real-world data. However, there are a certain number of assumptions to simplify the problem	.Yes., partly (UNHANDLE - time-linkage is mentioned in real-world scenarios (section VII, page 1219) but not considered in the context of the paper)	.Yes. (GA with different strategies)	.Partly.	.Yes. (.VISIBLE.)	.Yes.	.S.	Optimality only (Maximise income by appropriately choosing mixed grazing strategy)	The price of farm products changes linearly. This type of change is observed from real-world data, but no inflation was taken into account. There are recurrent changes. Some other changes in this problem follow problem-specific rules (Government subsidy reform and the dynamic of grouse population)	.Tracking. (.CLOSE. - The authors choose tracking based on the believe that it would be better when solving problem with gradual parameter change)	.Yes.	.No.	.No.	.Yes.	.Yes. (government subsidy policy will change in the future)
Dynamic Optimization of Fed-batch Fermentation Processes (Rocha *et al.* 2005)	Ecoli Fermentation process (Refers to reference [4] where it has been used for a real problem)	The problem here is to find an optimal control trajectory for the fermentation process. Note that this problem is solved in a combined way of both offline and online approaches. The offline phase is to create solution according to the known numerical problem while the online phase is to deal with any noise in real-time. In addition, the "dynamic" part (noise) is artificially generated. It is not clear if noises in real-world situations would follow the same distribution as the simulated problem.	.Yes. (UNKNOWN - it is not clear if the time-linkage property is taken into account during the optimisation process)	.Yes. (white-box EA which updates the population with the new values of parameters. EA does not handle change but process what users give it)	.N/I.	.Yes. (.VISIBLE. - changes need to be detected but they are detected by a separate sensor, not by the solver itself)	.Yes.	.S.	Optimality (Fermentation Productivity) & quick recovery (the algorithm is required to provide solutions after a certain period of time) & Reference-solution displacement restriction (online solutions need not to be very different from the reference solution).	It is reported that "several sources of noise can contribute to the changes in the observed values of the state values". However, the tested dynamics are artificially generated and might not reflect real-world situations.	.Tracking. (.QUICK. - when a change happens the EA takes the last population and adjust it to create new solutions. The purpose of tracking is to produce good solution in a short period of time)	.Yes.	.No.	.No.	.N/I.	.No.

Table 2 Continuous EA/metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / metaheuristics?	Predictable?	Visible	Constr. probs?	S/M-obj	Optimisation goal	Types of dynamics	Restart/ Track	Factors that change			
												Parameters of obj func	Domain range	Number of variables	Other prm Constraints
Dynamic optimization of watering mandarins (Morimoto *et al.* 2007)	Real data from Japan	The problem here is to increase the sugar content and decrease in citric acid content. It is not clear if the procedure described in diagram in Fig.1 is carried out only once (i.e. finish after one year) or will it be carried out over and over in an open-ending cycle. In the earlier case the problem is solved offline, and in the latter case it is possible to classify the problem as an online problem with time-window.	.Yes. (UNKNOWN - the output of the algorithm is the required amount of water, which could in turn affect the quality of mandarin in the future)	.Yes. (GA to find the optimal trajectory of the control parameter u for watering the mandarins. The dynamic model is formulated using a neural network based on historical data.)	.Partly. (Changes are periodical and seasonal)	.Yes. (.VISIBLE.)	.No.	.S. (Aggretated Objective Function)	Optimality (Increase the sugar content and decrease in citric acid content)	Changes are seasonal	.N/I. (Solved offline, may be restarted for the next cycle)	.Yes. (sunshine duration is time-dependent but its dynamic is known beforehand due to the way the problem is solved)	.No.	.No.	.No. .No.
Training a neural network to approximate the dynamic model of unmanned aerial vehicles (UAV) (Isaacs *et al.* 2008)	Fixed Wing Multi-Input Multi-Output Unmanned Aerial Vehicle system.	The problem here is to to minimise the learning error of the neural network, which is used to represent the dynamic model of UAV systems. The reason why the neural network needs to be trained online is because of the significance of environmental noises. These noises might make offline training ineffective.	.No.	.Partly. (Memetic Algorithm)	.No.	.No. (.INVISIBLE. - the algorithm detects changes by re-evaluating a random solution)	.N/I.	.M.	1. Quick recovery (solving and training time must be less than the sample time so that the algorithm can predict ahead of time) 2. Optimality (Train a NN with minimum error)	Changes are noises.	.Tracking. (.QUICK. - the reason is because the authors believe that the time needed to find new Pareto set from the previously converged Pareto set is smaller)	.Yes. (outputs of the UAV longitudinal system: pitch rate, forward velocity and vertical velocity)	.N/I.	.N/I.	.N/I. .N/I.
The Odor Source Localization problem (Jatmiko *et al.* 2008, Jatmiko *et al.* 2006)	Hardware (to be implemented) and Software Simulation Environment with Robots (although the environment is a lab-based one, the authors did link each property of the environment to properties that are commonly seen in real-life situations)	The problem here is to locate the source of chemical odour using mobile robots. The environment is changing overtime due to the wind and the diffusion of the odor.	.No.	.Yes. (Charged PSO)	.No.	.No. (.INVISIBLE.)	.No.	.S.	Optimality (Locate and move to the odour source) & Quick recovery (convergence time to reach a certain value is measured)	Odor dynamic is simulated using the Odor Gaussian Distribution, wind turbulence and chemical diffusion. Wind turbulence is chaotic, diffusion is non-linear and there are also sensor noises.	.Tracking. (.CLOSE. - due to the fact that a large part of the problem might still remain the same after the change and hence the new global optimum might not be far from the previous one)	.Yes.	.No.	.No.	.No. .No.

Table 2 Continuous EA/metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / metaheuristics?	Predictable	Visible	Constr. prob-lems?	S/M-obj	Optimisation goal	Types of dynamics	Restart/ Track	Factors that change: Parameters of obj func	Factors that change: Domain range	Factors that change: Number of variables	Other prm	Constraints
Adaptive contamination source identification (Liu *et al.* 2006, Liu 2008*b*)	Data from the U.S. Environmental Protection Agency and data of the virtual city Micropolis from Georgia Institute of Technology. The real-world data is used to represent the static network structure. The dynamic (contamination) is simulated artificially but it does follow the (simplified) hydraulic condition found in real-world cases	The problem here is to identify the origin of contamination in the water network. This case is an interesting example of how inverse modelling can be represented as dynamic optimisation problem! (the search space changes because the information about the problem is not fully known and can only be gradually revealed when time goes by)	.No.	.Yes. (multi-population GA-based approach)	.N/A.	.Yes. (.VISIBLE. - through a system of observers in the network)	.No.	.S.	Optimality (minimise the error between the predicted value and the observed value & choose the correct origin of contamination) & Quick recovery (it is mentioned that solutions need to be produced before new observations are received)	There are problem-specific changing rules: over time the observed concentration of sensors changes, revealing the necessary data to find the contamination origins.	.Tracking. multiple good solutions. The reason is because when time goes by one of those good solutions might become the optimal one (.CLOSE.).	.Yes. (dynamic parameters: observed concentration of sensors)	.No.	.No.	.N/I.	.N/I.
Multi-dimensional visual tracking (Pantrigo *et al.* 2008)	Two real-world case studies: "jump" and "run" were tested for articulated object tracking. In addition, the real-world video from CVBase'06 dataset were used to test multiple object tracking	The problem here is to to optimise the precision of the visual tracking system in tracking animation objects	.No.	Partly (Scatter Search (used with particle filter))	.Partly. (the pose of the geometric model for the next frame can be predicted)	.Yes. (.VISIBLE.)	.Yes.	.S.	Optimality (to optimise tracking precision) and Quick recovery	There are problem-specific changing rules: the type of dynamic depends on the type of object movement.	.Tracking. multiple solutions (.CLOSE. - to use previous knowledge because objects only move gradually;.LEARN. - the previous knowledge (previous positions of object) is also used to predict future changes)	.Yes.	.No.	.No. for the tested problem	.N/I.	.N/I.
Controller for a DSTATCOM in an electric ship power system (Mitra & Venayagamoorthy 2008)	A hardware setup (simplified model of the ship system) was tested in the lab	The problem here is to maintain regulation to a reference value for bus voltage in a ship power system. Here the deviation of voltage is kept minimum during the real-time process using AIS. This approach works because a robust solution has already been devised during the offline phase, and the task of AIS is to try to keep deviation from this robust solution to a minimum level	.N/I. (This property is not mentioned and is not taken into account by AIS during the optimisation process. The mathematical model of the control system is not provided so it is not clear whether and how the time-linkage property behaves in the system)	.Yes. (PSO to find the initial static solution, and AIS to adjust the solution online to minimise disturbances)	.N/I.	.No. (.INVISIBLE. - the solver in the DSP does not know about the errors (changes) beforehand. Signals will be sent to the solver and the solver has to adaptively deal with changes if there is)	.Yes.	.S.	Reference-solution displacement restriction (maintain regulation to a reference value for bus voltage); Spec Satisfaction (Stability - make sure that future solutions are within the allowable regulation range during the real-time process)	Changes are noises during the operation of the control system)	.Tracking. (.LEARN. - partly: the feedback rule does use information from the past (previous error) to calculate the new solution)	.N/I. (there is no detail about the mathematical model of the control system)	.N/I.	.N/I.	.N/I.	.N/I.

Table 2 Continuous EA/metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / meta-heuristics?	Predictable	Visible	Constr. prob-lems?	S/M-obj	Optimisation goal	Types of dynamics	Restart/ Track	Factors that change: Parameters of obj func	Factors that change: Domain range	Factors that change: Number of variables	Factors that change: Other prm	Constraints
Optimal visual proportional differential controller (Wang, Tao & Cho 2008)	Experiments with a physical system was done	The problem here is to minimise the deviation of the actual robot trajectory from a reference, pre-planned trajectory. The mathematical model has already been setup offline, but the optimisation process (parameter tuning) is assumingly be done online via experiments!	.Yes. (UNKNOWN - mainly offline because the mathematical model was calculated offline. It is not clear if during the offline process the time-linkage property is taken into account)	.Yes. (GA to minimise the errors of the control parameters)	.No. (the errors/ disturbances are not predictable)	.Yes. (.WINDOW. - At the beginning of a new time-window, the visual tracker send information about the current position of the object to create a new objective function for the GA to solve)	.Yes.	.S.	Reference-solution displacement restriction (minimise the deviation of the actual robot trajectory from a reference, pre-planned trajectory); Quick recovery;	Changes are linear	.N/I. (.UNKNOWN. - for each time step the new control values is calculated based on the previous solutions in the past. It is however not clear if the EA also follows the tracking approach or is restarted at each time step)	.Yes.	.N/I.	.No.	.N/I.	.Yes.
Evaporator system (Sonntag *et al.* 2008)	A rigorious simulation model of an evaporator system is developed in accordance with Bayer Technology Services	The problem here is to provide an optimal trajectory of states for an evaporator system. Note that the EA approach can only be applied in an offline way (the time and rule of switching are all known!).	.Yes.	.Offline. only (Lookahead minimisation is performed via nonlinear optmization and EA. The EA approach can only be applied in an offline way (the time and rule of switching are all known!))	.Partly. (the dynamic behaviour is predictable but the errors are not).	.Yes. (.WINDOW. - after being detected by sensors, changes are made visible to the optimiser at the beginning of each time window using the feedback mechanism)	.Yes.	.S.	Spec satisfaction	Changes are non-linear	.Tracking. (.LEARN. - the principle of predictive control requires that at each receding horizon solutions from the past are used to predict the future behaviour.).	.Yes. (the dynamical system also switches between different modes)	.No.	.Yes. (Combination of binary and real variables means that when the system switches mode, the number of variables might change)	.N/I.	.Yes.

Table 3: Combinatorial real-world references that use non-metaheuristic methods

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / meta-heuristics?	Predictable	Visible	Constr. prob-lems?	S/M-obj	Optimisation goal	Types of dy-namics	Restart/Track	Factors that change: Parameters of obj func	Domain range	Number of vari-ables	Other prm	Constraints
Air traffic control in ter-minal areas (Bianco *et al.* 2006)	Real-data from two airports in Milan and Rome	The problem here is to find the optimal land-ing/taking off schedules for all airplanes in the air-port in real-time to meet the specifications, time limit and requirements of the airport. Note that the time limit in this type of problems is usually mod-elled as a specific type of constraints.	.Yes. (Decision of which aircraft to land/takeoff first would change the problem in the next time step. However the property is not fully taken into account (HANDLED & UNHANDLE). There is also the previous-solution dis-placement restriction)	.No., solved by heuris-tics	.No.	.Yes. (.VISI-BLE. - the system is informed whenever a new aircraft arrives)	.Yes. (e.g. the specifi-cation in land-ing/taking off of each type of aircraft and the require-ments in rese-quenc-ing air-crafts)	.S.	1: quick recov-ery (to meet the limit of the time win-dow - less than one second); 2: Previous-solution dis-placement restriction & Optimality (these two are all integrated in the obj func); Spec Satisfac-tion: each type of airplane has a specification in landing/taking off.	There are non-linear changes caused by the increase of throughputs. There are also problem-specific chang-ing rules, which are controlled by air regula-tions and other restrictions	.Tracking. (.DISP. - The new schedule is adjusted from the previous one. There are heuristic rules to keep certain aircraft positions almost fixed and to prevent aircraft to shift position too much.)	.Yes.	.No.	.Yes.	.No.	.No.
Optimal ambu-lance loca-tion in urban areas (Ingolfsson *et al.* 2008)	The use of the proposed model (for cities with one million popula-tion) is illustrated using the real-world data of Ed-monton. Data from three real-world ambu-lation location projects is also used for inves-tigation	The problem here is to find the optimal ambu-lance locations in urban areas and adjust the solutions to cope with changes. The problem is solved in both offline and online ways: Offline for each time window (knowledge about the dynamics from previous data is incorporate into the algorithm.); and online in the long run (The authors did mention the case of solving the problem online when de-mand varies, using time-window approach with 168h for each window). It should be noted that in the proposed model, the arrival rate of calls is artificially modelled using Poisson distribution. This distribution however is widely believed to ac-curately represent the distribution of real-world calls.	.N/I. (No men-tion to the time-linkage property. However, hypo-thetically that can happen. For example the quality of current services might affect the demand in the future etc)	.No., the authors use heuris-tics + branch-and-bound methods	.Partly.&.Yes. (.Yes. for the changes in each time win-dow;.N/I. for de-mand change)	.Yes. (a) Un-certainty in each time window: solved offline (b) De-mand: pseudo-visible (.WIN-DOW. - after each time-window)	.Yes. (max-imum num-ber of ambu-lance)	.S.	Optimality (op-timal ambulance deployment to maximise the coverage given the number of ambulance available)	Emprical data of pre-travel delays follows a lognormal distribution	.Restart. (ap-plicable because the period be-tween each time-window is long)	.Yes. (changes in the travel time (speed of vehicles; routes etc), delays prior to the trip, changes in the availabil-ity of ambu-lance and changes in de-mand	.No.	.No. (the num-ber of sta-tion is fixed)	.N/I.	.No.

Table 3 Combinatorial non-metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / meta-heuristics?	Predictable	Visible	Constr. problems?	S/M-obj	Optimisation goal	Types of dynamics	Restart/Track	Factors that change				Constraints
												Parameters of obj func	Domain range	Number of variables	Other prm	
Airlift mission monitoring and dynamic rescheduling (Wilkins *et al.* 2008)	Actual (full-sclae) schedules from the USAF Air Mobility Command and simulated data feeds by the Airforce Research Laboratory were claimed to be "as similar to actual data feeds as possible".	The problem here is to effectively schedule and reschedule the airlift missions of the US Air Force under changing environments. The proposed system is a decision-support system which comprises multiple components to do different tasks as monitoring changes; evaluating costs; finding optimal rescheduling options; and interacting with users. The system is still under development/testing and has not been fully integrated into action yet.	.Yes. (HANDLED - it is recognised that rescheduling options have potential impacts on future operations, and that it is necessary to take into acount this time-linkage property (impacts on future operations) to solve the problem effectively	.N/I.	.Partly. (the consequence of a rescheduling decision can be predicted. Environmental changes cannot be predicted)	.Yes. (.VISIBLE. - changes need to be detected automatically but this is not the task of the optimiser (the Scheduler) because there is a separate component to monitor/detect changes)	.Yes. (mission's requirements, resource availability , usage constraints and operation regulations)	.N/I.	Optimality; Previous-solution displacement restriction (to minimise disruption to other actions); Quick recovery; Spec Satisfaction: Each mission might have a restricted time frame to fulfill.	.N/I.	.Tracking. (.DISP. - to minimise disruption to other actions)	.N/I. (there is no detailed mathematical description of the objective function)	.N/I.	.N/I.	.N/I.	.Yes. (mission's requirements, resource availability and usage constraints can all change over time. In addition, in certain cases constraints can be relaxed (i.e. some constraints are removed) so that some missions can be executable in time)
Managing restaurant tables using constraints (Vidotto *et al.* 2007)	a medium-size restaurant in Douglas, Cork City, Ireland	The table management problem is modelled as a dynamic scheduling problem , where tables are resources and parties are tasks (with start/end times and a size). Parties are modelled as decision variables and tables are values to be assigned.	.Yes. (HANDLED - it is recognised that the current booking/planning decision will affect the future problem (future coverage of tables).) The authors also try to handle the time-linkage feature by predicting the future coverage of the current tables given the current booking.	.No. (the problem is solved using some heuristics)	.Partly. (the consequence of a booking decision can be partly predicted. Other changes (future booking requests, future walk-in customers etc) cannot be predicted)	.Yes. (.VISIBLE.)	.Yes.	.S.	Quick recovery (there is a limit in the time to find a solution. Any solution that takes longer than the limit will not be accepted); Previous-solution displacement restriction (disruption must be minimised by not only limiting the amount of changes to previous solution but also limiting the number of changes); Optimality; Spec Satisfaction (schedules made now need to be ensured not conflict with the allocation and time of other pre-booked schedules)	.N/I.	.Tracking. (.DISP. - to minimise disruption to other actions)	.Yes. (many factors can change, e.g. the number of people in each party and the starting and ending time of each party)	.No. (number of tables and types of tables are fixed)	.Yes. (the number of parties changes over time)	.N/I.	.Yes. (some constraints changes due to the change in number of variables)

Table 3 Combinatorial non-metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / metaheuristics?	Predictable	Visible	Constr. problems?	S/M-obj	Optimisation goal	Types of dynamics	Restart/Track	Factors that change				
												Parameters of obj func	Domain range	Number of variables	Other prm	Constraints
Dynamic Channel Assignment in Wireless LANs (Wang, Wu & Liu 2008)	A 500m2 testbed for the wireless network was setup in the office, under different changes in the environment	The problem here is to efficiently adapts the channel assignments in the wireless network to deal with variations in traffic. It should be noted that this application is an example where not all changes are reacted. Only "significant changes" are considered.	.N/I. (Not mentioned)	.No. (a semi-definite programming relaxation technique is used)	.N/I.	.Yes. (.WINDOW. - the optimisation process is divided into equal time-windows. However, the solver still needs a separate component to evaluate the magnitude of changes to determine if it is necessary to react to changes)	.Yes. (but not explicitly specified in the paper)	.S.	Optimality; Quick recovery (the solution need to be delivered before the end of the time window)	.N/I. (it is observed that trafic is "often not uniform among channels". However, the tested dynamics are artificially generated)	.Restart. (there is no explanation why restarting is chosen over tracking in this lab-based experiment. It should be noted that restart is carried out only if changes are considered "significant")	.N/I. (there is no detailed description of the objective function and decision variables)	.N/I. (there is no detailed description of the objective function and decision variables)	.N/I. (there is no detailed description of the objective function and decision variables)	.N/I. (there is no detailed description of the objective function and decision variables)	.N/I. (there is no detailed description of the objective function and decision variables)
Dynamic assignment in P2P networks (Martinez *et al.* 2008)	The dynamics of the peer connection/ disconnection is based on a the logs of user's behaviour of a live-video service of a medium-size ISP	The problem here is a dynamic assignment problem in a P2P network designed for sending real-time video over the Internet in a highly dynamic environment where connections and disconnections occur frequently. The purpose is to periodically eassigning network connections to maximise the global expected Quality-of-Experience to clients.	.Yes. (HANDLED - it is recognised that the current assignment might influence the appearance of connections and disconnections in the future. This feature is taken into account and is used as criteria to evaluate the solution quality (Subsection 3.1). However, there is no detail about how this procedure is implemented)	.No. (solved by a GRASP metaheuristic)	.Partly. (it is argued that the connection/disconnection behaviours of a particular client is predictable and the proposed method rely on this assumption to improve the performance)	.Yes. (.WINDOW. - the optimisation process is divided into fixed time-windows)	.Yes. (there are restrictions for the upstream and downstream bandwidths of nodes and servers. There are also other network-related constraints)	.S.	Optimality (it is not clear if other optimisation goals are also taken into account)	.N/I.	.Tracking. (at the beginning of each time-window the GRASP heuristics is started from the previous found solution. Newly connected and newly disconnected nodes will be added to / removed from the existing solution. The authors do not clearly state why the tracking approach is chosen. However, it can be assumed that tracking is the only viable option because otherwise all existing connections will be reset -.DISP.)	.N/I. (there is not enough details to for us to decide if this factor changes)	.N/I.	.N/I. (there is not enough details to for us to decide if this factor changes)	.N/I. (there is not enough details to for us to decide if this factor changes)	.N/I. (there is not enough details to for us to decide if this factor changes)

Table 3 Combinatorial non-metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / meta-heuristics?	Predictable	Visible	Constr. problems?	S/M-obj	Optimisation goal	Types of dynamics	Restart/Track	Factors that change: Parameters of obj func	Factors that change: Domain range	Factors that change: Number of variables	Factors that change: Other prm	Factors that change: Constraints
Optimal load balancing (Soga *et al.* 2008)	Experiments were done in a real-life computing cluster in Japan	The problem here is to dynamically adjust the implementation of Broadcast operation, one of the most popular collective communications in parallel applications to avoid waiting time in parallel processes. The dynamic in this problem is caused by the fact that although theoretically all parallel processes should begin their tasks at the same time, in reality due to the imbalance of workload of each process, processes may start their operations at different times. Because of that, it might be necessary to re-order the processes to minimise the wait time caused by under-load processes.	.N/I.	.No. (solved by a proposed heuristic)	.No.	.No. (.INVISIBLE. The changes (load-imbalance of the system) is detected by measuring the wait time of receive operations in Broadcasts.). It also means that changes cannot be detected just by re-evaluating a few solution	.N/I. (no detail of the objective and constraint functions is provided)	.N/I. (no detail of the objective and constraint functions is provided)	Optimality (minimise the wait time caused by load-imbalance)	.N/I. (dynamic data are generated artificially)	.Tracking. (the order of some messages in the Broadcast will be adjusted to handle the change. The authors do not clearly state why the tracking approach is chosen. However, it can be assumed that tracking is the only viable option because otherwise all existing parallel tasks will be terminated -.DISP.)	.N/I. (no detail of the objective and constraint functions is provided)	.N/I. (no detail of the objective and constraint functions is provided)	.N/I. (no detail of the objective and constraint functions is provided)	.N/I. (no detail of the objective and constraint functions is provided)	.N/I. (no detail of the objective and constraint functions is provided)
Path-finding for intelligent agents in AI games (Bulitko *et al.* 2007)	Maps from real commercial games	The problem here is to cumulatively build a path for an intelligent agent (in real-time AI games) to travel from one place to another when time goes by. In this type of problems (path-finding), the search space (terrain map) is initially unknown to the search agent. Over time, when the search agent moves around the landscape, it will realise more and more about the landscape. Because at each time step the search agent is restricted to discover only a limited area of the search space, we can consider the problem as a dynamic problem in which at each time step the algorithm need to deal with a slightly different objective function. This is an example of cases where although the objective function is completely known, it is too expensive to solve offline or there is some strict rules preventing it from being solve offline, and hence it needs to be solved online	.Yes. (in the sense that the outcome of the algorithm at one time-window will decide the starting point (and hence the search space) of the algorithm in the next time-window. The time-linkage property is handled in the sense that at the current time the solver plans n actions away into the future. HANDLED)	.No. (solved by a special heuristics)	.Partly. (a small segment of the map can be predictable based on current knowledge)	.Yes. (.WINDOW. - because the optimisation process is divided into fixed time-windows)	.Yes.	.M. (The single-objective case was also tested)	Optimality; Quick recovery (there is a time-per-action limit in computer games regardless of problem size); Spec Satisfaction (the search agent needs to eventually reach a goal/ destination)	.N/I.	.Tracking. (.LEARN. - the search agent re-uses (and starts from) the knowledge that it has learnt from the past to initiate its search at the beginning of each new time step. The purpose of tracking might not be that the global optimum of the new problem is close to the old one, but that tracking enable the continuity of the path-finding approach and also to learn more about the surrounding map. It may also facilitate producing a solution more quickly (..QUICK..))	.Yes.	.Yes.	.No. (the number of actions in one time-window is fixed)	.Yes.	.Yes.

Table 4: Combinatorial non-metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / metaheuristics?	Predictable	Visible	Constr. problems?	S/M-obj	Optimisation goal	Types of dynamics	Restart/Track	Factors that change: Parameters of obj func	Factors that change: Domain range	Factors that change: Number of variables	Factors that change: Other prm	Constraints
Continuous field path-planning for robots (Mills-Tettey *et al.* 2008)	Two real-world robots were tested	The problem here is to cumulatively build an optimal path for a robot to move from one place to another when time goes by. An offline path might have already been set up, but because the actual terrain might be different from the pre-planned path due to unknown obstacles, the robot needs to adjust its path to get the destination. Because of the unknown factors along the path and because of the fact that the robot needs to find the actual path online, the problem is a dynamic optimisation problem.	.Yes. (in the sense that the outcome of the algorithm at one time-step will decide the starting point, energy, and storage data (and hence the objective function and constraints) of the robot in the next time-step. The time-linkage property is handled in the sense that the restriction of robot movement is relaxed to ensure that its future path can lead to the destination given the current level of energy - HANDLED)	.No.	.Partly. (some properties, e.g. the dominance of a cell compared to another, can be predicted)	.Yes. (.VISIBLE. - the robot needs to detect changes itself but it uses dedicated sensors for this purpose. The solver does not need to detect changes)	.Yes.	.S.	Optimality (minimising traversal cost); Quick recovery (there is a time-per-action limit for the robot); Spec Satisfaction (the robot needs to eventually reach a destination position); Reference-solution displacement restriction (there might also be a pre-planned path that the robot should follow as close as possible)	.N/I.	.Tracking. (to get a new solution quickly (.QUICK.) and to use knowledge from the past to learn more about the environment and how to handle the environment (.LEARN.))	.Yes. (at each time-step when the robot moves to a new place, its sensor may discover some discrepancies between this new environment and the planned map. In such case the objective function needs to be adjusted)	.N/I.	.N/I.	.N/I.	.Yes. (the constraints will also change because the total available energy and the data storage also changes over time depending on the position of the robot and the time the robot takes to get to that position)
State estimation in three-tank system (Pina & Botto 2008)	The experiment was done in a AMIRA DTS200 three-tanks system	This reference does not focus on optimisation but on estimating the unknown state of hybrid systems. In this hybrid system the number of possible discrete modes is known.	.Yes. (HANDLED - the current control value will determine how the current dynamic system will be in the future. In addition, the algorithm also determine when and how a switch-mode should happen in the future)	.No.	.Partly. (the time and the way a switch occurs is determined by the controller, but the errors/ disturbances are not predictable)	.Yes. (.WINDOW. - the optimisation process is divided into time-windows and hence changes are assumed to occur at the beginning of each time window)	.Yes.	.S.	.N/A. (state estimation, not optimisation)	Changes are non-linear	.N/A. (because the goal of the research is to estimate the state, not to find optimal solution)	.Yes. (the objective function also switches between different modes)	.N/I.	.N/I.	.N/I.	.Yes.

Table 4 Combinatorial non-metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / meta-heuristics?	Predictable	Visible	Constr. problems?	S/M-obj	Optimisation goal	Types of dynamics	Restart/Track	Factors that change				
												Parameters of obj func	Domain range	Number of variables	Other prm	Constraints
Bus Scheduling in London (Andrews & Tuson 2005)	Bus scheduling in London, interview from practitioners	Dynamic bus scheduling problem	.N/I.	.N/I.	.Partly. (the level of predictability is 'medium')	.Yes. (.VISIBLE.)	.Yes.	.M. (the service needs to meet the standard and the operating cost needs to be optimised)	1: Optimality; 2: Previous-solution displacement restriction; 3: Reliability (.Other. goal)	Changes are stochastic but it is not clear if it follows any distribution due to the lack of information. Changes are also periodical and the level of periodicity is variable.	.N/I.	.Yes.	.N/I.	.N/I.	.N/I.	.Yes. (variable in amount of resources; changes in the route map, changes in travelling time)
Courier Service in London (Andrews & Tuson 2005)	Courier service in London, interview from practitioners	Dynamic scheduling for courier service in London	.N/I.	.No.! (solved by heuristic and neural network)	.Partly. (the level of predictability is 'medium')	.N/I.	.N/I.	.N/I.	quick recovery is the most concern (It is required that the algorithm need to recover quickly to a 'minimum standard' before any improvement can be made)	.N/I.	.N/I.	.N/I.	.N/I.	.N/I.	.N/I.	.N/I.
Peptide Identification (Andrews & Tuson 2005)	Interview from practitioners	Peptide identification problem	.N/I.	.N/I.	.Partly. (the level of predictability is 'high')	.No. (.UNKNOWN. -it is not clear if the algorithm needs to detect changes or the time-window approach can be used to handle changes)	.N/I.	.N/I.	.N/I.	Changes are noises and the level of periodicity is "high".	.N/I.	.Yes. (noisy obj function)	.N/I.	.Yes.	.N/I.	.N/I.
Communication Middleware (Andrews & Tuson 2005)	Interview from practitioners	Communication middleware problem	.N/I.	.N/I.	.Partly. (the level of predictability is 'low')	.No. (.UNKNOWN. -it is not clear if the algorithm needs to detect changes or the time-window approach can be used to handle changes)	.N/I.	.N/I.	.N/I.	Some changes are periodical and the level of periodicity is 'low'	.N/I.	.N/I.	.Yes.	.Yes.	.N/I.	.N/I.

Table 4 Combinatorial non-metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / metaheuristics?	Predictable	Visible	Constr. problems?	S/M-obj	Optimisation goal	Types of dynamics	Restart/Track	Factors that change				
												Parameters of obj func	Domain range	Number of variables	Other prm	Constraints
Dynamic Resource Allocation (Andrews & Tuson 2005)	Interview from practitioners	Dynamic resource allocation problem	.N/I.	.N/I.	.No.	.No. (.UNKNOWN. -it is not clear if the algorithm needs to detect changes or the time-window approach can be used to handle changes)	.Yes.	.N/I.	1: Optimality & quick recovery; 3: Previous-solution displacement restriction	Some changes are periodical and the level of periodicity is 'low'	.N/I.	.Yes.	.No.	.Yes. (rarely)	.N/I.	.Yes. (available resources)
Dynamic Scheduling (Andrews & Tuson 2005)	Interview from practitioners	Dynamic scheduling problem	.N/I.	.N/I.	.No.	.No. (.UNKNOWN. -it is not clear if the algorithm needs to detect changes or the time-window approach can be used to handle changes)	.Yes.	.N/I.	1: Optimality & quick recovery; 3: Previous-solution displacement restriction	Some changes are periodical and the level of periodicity is 'low'	.N/I.	.Yes. (fitness function changes as the priorities change and the tolerant amount changes)	.No.	.Yes. (rarely)	.N/I.	.Yes. ('number and type of scheduling elements involved')
Risk Minimization Problem (Andrews & Tuson 2005)	Interview from practitioners	Risk Minimisation Problem	.N/I.	.N/I.	.No.	.No. (.UNKNOWN. -it is not clear if the algorithm needs to detect changes or the time-window approach can be used to handle changes)	.Yes.	.N/I.	1: Optimality & quick recovery; 3: Previous-solution displacement restriction	Some changes are periodical and the level of periodicity is 'low'	.N/I.	.Yes. (fitness function changes as the risk types and risk assessments change)	.No.	.Yes. (rarely)	.N/I.	.Yes.
Travel time in ambulance deployment and station location problems (Budge *et al.* 2008)	Administrative data for one year of high priority calls (7457 calls) in Calgary, Alberta.	This research does not solve the ambulance deployment and station localtion problem but instead reveals the real-world dynamic properties of the problem. The statistical analysis of history data can be used to assist the planning of ambulance services more effectively	.N/A.	.N/A.	.Partly.	.N/A.	.N/A.	.N/A.	.N/A.	Changes follow a leptokurtic distribution with a coefficient of variation that decreases with distance. Changes also depend on problem-specific rules such as the time of day and travel distances.	.N/A.	.Yes. (changes in the speed of vehicles; changes in route;)	.N/A.	.N/A.	.Yes. (changes in the speed of vehicles; changes in route;)	.N/A.

Table 5: Continuous real-world references that use non-metaheuristic methods

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / metaheuristics?	Predictable	Visible	Constr. probs?	S/M-obj	Optimisation goal	Types of dynamics	Restart/Track	Factors that change: Parameters of obj func	Factors that change: Domain range	Factors that change: Number of variables	Factors that change: Other prm	Constraints
Parameter estimation in Polymerisation process (Prata *et al.* 2006)	Dynamic data was observed from a real-life industrial bulk propylene polymerisation process for six months		.Yes., partly (HANDLED - although not directly, current estimation result does influence the value of the chosen control variable for next step, which in turn influence the problem in the next step.)	.No. (special prm estimation technique is used and is restarted after each time window)	.No. (the errors/ disturbances cannot be predicted)	.Yes. (.WINDOW. - the optimisation process is divided into time-windows and hence changes are assumed to occur at the beginning of each time window)	.Yes.	.S.	Reference-solution displacement restriction (minimise errors between the reference trajectory and the observed solution.); Quick recovery (the algorithm is required to provide solution in a sorter time than the sampling period)	The measurement fluctuations follow a normal distribution with zero means. The dynamic system also changes non-linearly.	.N/I.	.Yes. There are two types of dynamics in the objective function. The first is data error at each time window. The second is the dynamic of the control system	.No.	.No.	.N/I.	.Yes.
Building demand control problem (Sane & Guay 2008)	Building construction is taken from real-world examples, solar loads and ambient conditions are achieved from the TMY database for Hartford, CT in the months of July, load schedule are chosen to correspond to typical office application		.Yes. (HANDLED - the considered plant controls the room temperature and the performance of other related equipments, which in turn influences the dynamic of the problem in the future)	.No. (a special MPC technique is used)	.N/I.	.Yes. (.VISIBLE. - environmental changes are visible, and control changes are controlled by the algorithm)	.Yes.	.S.	Spec Satisfaction (maintain stability) Optimality (minimise the electric utility cost);	Some changes are linear. The data also show that some changes (outside air temperature and solar irradiation) have a periodical property.	.Tracking. (.LEARN. - the principle of predictive control requires that at each receding horizon solutions from the past are used to predict the future behaviour)	.Yes.	.No.	.No.	.N/I.	.Yes.
Minimum-time travel for a vehicle with acceleration limits (Velenis & Tsiotras 2008)	The proposed method was applied to an actual road track to generate the velocity profile of a Silverstone F1 circuit	One interesting note in this research is that the size of the time-window is also optimised to maintain stability.	.Yes. (HANDLED - the current value of control variable would influence the future dynamic of the system)	.No. (a receding horizon approach is used)	.Partly. (the future dynamic can be predicted, but the errors/ disturbances are not)	.Yes. (.WINDOW. - the optimisation process is divided into time-windows and hence changes are assumed to occur at the beginning of each time window)	.Yes.	.S.	Spec Satisfaction (Stability - make sure that the speed will not exceed the "critical value" at any time step and there is an "escape plan" at the end of each time window); Optimality (minimise lap time of the F1 car); Quick recovery (the computation time needs to be short to finish within the current time window)	.N/I.	.N/I.	.Yes.	.No.	.N/I.	.N/I.	.Yes.

Table 5 Continuous non-metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / meta-heuristics?	Predictable	Visible	Constr. prob-lems?	S/M-obj	Optimisation goal	Types of dy-namics	Restart/Track	Factors that change			
												Parameters of obj func	Domain range	Number of variables	Other prm Constraints
Control DC-DC con-verters based on combi-natory opti-miza-tion (Ahmad & Liu 2008)	Only simulation was shown but the authors claim that the method has been ap-plied directly to hardware equipments		.Yes. (HAN-DLED - the current control value will deter-mine how the current dynamic system will be in the future. In addition, the algorithm also determine when and how a switch-mode event should happen in the future)	.No.	.Partly. ("all pos-sible state trajectories can be predicted on-line since the number of modes is finite, the dynam-ics is affine and au-tonomous ", but the errors/ dis-turbances are not)	.Yes. (.WIN-DOW. - Al-though distur-bances changes are not visible, the optimisation process is di-vided into time-windows and hence changes are assumed to occur at the beginning of each time window)	.No.	.S.	Reference-solution displacement re-striction (minimise the deviation of output voltage from a reference value)	.N/I. (the environmen-tal changes were generated artificially)	.Tracking. (.LEARN. - The past knowledge is used by the MPC to predict future behaviour of the system)	.Yes. (the dy-namical system also switches between different modes)	.No.	.No.	.N/I. .N/A.
Constant-pressure water supply system (Zhang & Li 2007)	Industrial constant-pressure water supply system	Nonlinear Model pre-dictive control	.Yes.	.No.	.Partly. (the dy-namic behaviour is pre-dictable but the errors are not).	.Yes. (.WIN-DOW. - after being detected by sensors, changes are made visible to the optimiser at the beginning of each time window using the feedback mechanism)	.Yes.	.S.	Reference-displacement restriction; and Spec Satisfaction (Stability (con-trol the plant for desired out-put) and other specifications)	The dynamics of the system are non-linear. There is no detailed infor-mation about the types of other changes..	.Tracking. (.LEARN. - the principle of predic-tive control requires that at each reced-ing horizon solutions from the past are used to predict the future behaviour. In addition the NLP optimiser also starts from the past solution).	.Yes.	.No.	.Yes. (The number of such variables of the op-timization problem at each step would change de-pending on how mant number of past inputs and outputs is necessary.)	.N/I. .Yes.
A contin-uous stirred tank heater mod-elling (Thornhill *et al.* 2008)	a continuous stirred tank heater pilot plant at the University of Alberta is used to validate the model	Simulation of a real system for teaching.	.Yes.	.No.	.Partly. (the dy-namic behaviour is pre-dictable but the errors are not).	.Yes. (.WIN-DOW. - after being detected by sensors, changes are made visible to the optimiser at the beginning of each time window using the feedback mechanism)	.Yes.	.S.	Reference-displacement restriction; and Spec Satisfaction (Stability (plant control) and other specifications)	The dynamics of the system are non-linear. There is no detailed infor-mation about the types of other changes.	.Tracking. (.LEARN. - the principle of predic-tive control requires that at each reced-ing horizon solutions from the past are used to predict the future behaviour. In addition the NLP optimiser also starts from the past solution).	.Yes.	.No.	.No.	.N/I. .Yes.

Table 5 Continuous non-metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / meta-heuristics?	Predictable	Visible	Constr. prob-lems?	S/M-obj	Optimisation goal	Types of dy-namics	Restart/Track	Factors that change: Parameters of obj func	Factors that change: Domain range	Factors that change: Number of variables	Factors that change: Other prm	Constraints
Heater-mixer setup (Srinivasarao *et al.* 2007)	a heater-mixer setup developed at the Department of Chemical Engineering, I.I.T. Bombay is used to validate the model	Simulation of continuous fermentation benchmark problem.	.Yes.	.No.	.Partly. (the dynamic behaviour is predictable but the errors are not).	.Yes. (.WINDOW. - after being detected by sensors, changes are made visible to the optimiser at the beginning of each time window using the feedback mechanism)	.Yes.	.S.	Reference-displacement restriction; and Spec Satisfaction (Stability (plant control) and other specifications)	The dynamics of the system are non-linear. There are also noises caused by errors and disturbances.	.Tracking. (.LEARN. - the principle of predictive control requires that at each receding horizon solutions from the past are used to predict the future behaviour. In addition the NLP optimiser also starts from the past solution).	.Yes.	.No.	.No.	.N/I.	.Yes.
Industrial fermentation process (Yu *et al.* 2006)	Real data modeling of a three-input three-output chemical process rig is used to evaluate the model	Experimental rig	.Yes.	No	.Partly. (the dynamic behaviour is predictable but the errors are not).	.Yes. (.WINDOW. - after being detected by sensors, changes are made visible to the optimiser at the beginning of each time window using the feedback mechanism)	.Yes.	.S.	Reference-displacement restriction; and Spec Satisfaction (Stability (plant control) and other specifications)	The dynamics of the system are non-linear. There is no detailed information about the types of other changes.	.Tracking. (.LEARN. - the principle of predictive control requires that at each receding horizon solutions from the past are used to predict the future behaviour.).	.Yes.	.No.	.Yes. (Yes as the structure of the regressor changes i.e. number of past inputs and outputs could change.)	.N/I.	.Yes.
Control of Residential Energy Resources (Houwing *et al.* 2007)	Residential electricity and aggregated heat demand data in 2006 from EnergieNed, the Dutch Federation of Energy Companies is used as input for the model.		.Yes.	.No.	.Partly. (the dynamic behaviour is predictable but the errors are not; the buying price of electricity is known a day in advance).	.Yes. (.WINDOW. - after being detected by sensors, changes are made visible to the optimiser at the beginning of each time window using the feedback mechanism)	.Yes.	.S.	Optimality (Household energy sourcing for minimum cost)	The dynamics of the system are linear. There is no detailed information about the types of other changes.	.Tracking. (.LEARN. - the principle of predictive control requires that at each receding horizon solutions from the past are used to predict the future behaviour. In addition the MILP optimiser might also start from the past solution).	.Yes. (the dynamical system also switches between different modes)	.No.	.Yes. (Combination of binary and real variables means that when the system switches mode, the number of variables might change?)	.N/I.	.Yes.
Solar Air Conditioning Plant (Menchinelli & Bemporad 2008)	Tested in a solar air conditioning plant in University of Seville, Spain.	Hybrid systems controlling the operating modes of the solar controller	.Yes.	.No.	.Partly. (the dynamic behaviour is predictable but the errors are not).	.Yes. (.WINDOW. - after being detected by sensors, changes are made visible to the optimiser at the beginning of each time window using the feedback mechanism)	.Yes.	.S.	Spec Satisfaction (Closed loop stability and other specifications); Robustness of the approach (.Other. Goal)	The dynamics of the system are non-linear. There is no detailed information about the types of other changes.	.Tracking. (.LEARN. - the principle of predictive control requires that at each receding horizon solutions from the past are used to predict the future behaviour.). The optimisation method is Mixed Integer Quadratic Programming	.Yes. (the dynamical system also switches between different modes)	.No. (it should be noted that binary variables are also involved)	.Yes. (Combination of binary and real variables)	.N/I.	.Yes.

Table 5 Continuous non-metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / meta-heuristics?	Predictable	Visible	Constr. prob-lems?	S/M-obj	Optimisation goal	Types of dy-namics	Restart/Track	Factors that change			
												Parameters of obj func	Domain range	Number of variables	Other prm Constraints
Control of DC-DC Switched Mode Power Sup-plies (Beccuti *et al.* 2009)	An integrated DC-DC con-verter through a fixed-point DSP is devel-oped to validate the model	Hybrid systems with model switching. The problem is not solved totally on-line. The problem is pre-solved off-line. The on-line part is to search in the result-ing look-up table	.Yes.	.No.	.Partly. (the dy-namic behaviour is pre-dictable but the errors are not).	.N/A. The re-gressor is trained using off-line data.	.Yes.	.S.	Spec Satisfaction (Closed loop sta-bility and other specifications); Robustness of the approach (.Other. Goal)	The dynamics of the system are non-linear. There is no detailed infor-mation about the types of other changes.	.N.A. Track op-timum is used in the off-line phase.Tracking is not choosen in online implementa-tion. For training the model off-line, receeding approach.	.Yes. (the dy-namical system also switches between different modes)	.No.	.Yes. (Combi-nation of binary and real variables)	.N/I. .Yes.
Control of two-stage matrix con-verter (Mariéthoz *et al.* 2008)	A laboratory prototype is developed to validate the model	Hybrid sys-tems with alternative toplogy switching	.Yes.	.No.	.Partly. (the dy-namic behaviour is pre-dictable but the errors are not).	.Yes. (.WIN-DOW. - after being detected by sensors, changes are made visible to the optimiser at the beginning of each time window using the feedback mechanism)	.Yes.	.S.	Spec satisfaction	The dynamics of the system are non-linear. There are also noises caused by errors and disturbances.	.Tracking. (.LEARN. - the principle of predic-tive control requires that at each reced-ing horizon solutions from the past are used to predict the future behaviour.).	.Yes. (the dy-namical system also switches between different modes)	.No.	.N/I.	.N/I. .Yes.
Heat Ex-change Re-actor (Haugwitz *et al.* 2007)	The experiment is done in a real chemical reac-tor, the Open Plate Reactor, developed by Alfa Laval AB.	High rate of conversion and temper-ature control needs to be achieved	.Yes.	.No.	.Partly. (the dy-namic behaviour is pre-dictable but the errors are not).	.Yes. (.WIN-DOW. - after being detected by sensors, changes are made visible to the optimiser at the beginning of each time window using the feedback mechanism)	.Yes.	.S.	Spec Satisfaction (Stability and other specifica-tions); Robustness (.Other. Goal); Safe, High rate of conversion within limits of reac-tion temperatures (.Other. Goal)	The dynamics of the system are non-linear. There are also noises caused by errors and disturbances.	.Tracking. (.LEARN. - the principle of predic-tive control requires that at each reced-ing horizon solutions from the past are used to predict the future behaviour.).	.Yes.	.No.	.No.	.N/I. .Yes.

Table 5 Continuous non-metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / metaheuristics?	Predictable	Visible	Constr. probs?	S/M-obj	Optimisation goal	Types of dynamics	Restart/Track	Factors that change: Parameters of obj func	Factors that change: Domain range	Factors that change: Number of variables	Factors that change: Other prm	Constraints
Polymer Electrolyte Membrane fuel cell system (Fiacchini *et al.* 2008)	Use a dynamic model, which was validated in a real plant in (del Real *et al.* 2007)	Validated against a real model data	.Yes.	.No.	.Partly. (the dynamic behaviour is predictable but the errors are not).	.Yes. (.WINDOW. - after being detected by sensors, changes are made visible to the optimiser at the beginning of each time window using the feedback mechanism. The algorithm can also control some changes (model switching). The model switching happens when the operating ranges is different)	.Yes.	.S.	Spec satisfaction (Safe region of operation)	The dynamics of the system are non-linear.	.Tracking. (.LEARN. - the principle of predictive control requires that at each receding horizon solutions from the past are used to predict the future behaviour.).	.Yes. (the dynamical system also switches between different modes)	.No.	.Yes. (Combination of binary and real variables means that when the system switches mode, the number of variables might change)	.N/I.	.Yes.
Control of a sugar factory (de Prada *et al.* 2008)	Use a full scale simulator of a real sugar factory. The simulator is described in (Merino *et al.* 2006)	Validated against a real model data	.Yes.	.No.	.Partly. (the dynamic behaviour is predictable but the errors are not).	.Yes. (.WINDOW. - after being detected by sensors, changes are made visible to the optimiser at the beginning of each time window using the feedback mechanism. The algorithm can also control some changes (model switching))	.Yes.	.S.	Optimality (continuous control and batch units scheduling)	The dynamics of the system are non-linear.	.Tracking. (.LEARN. - the principle of predictive control requires that at each receding horizon solutions from the past are used to predict the future behaviour.).	.Yes. (the dynamical system also switches between different modes)	.No.	.Yes. (Combination of integer and real variables means that when the system switches mode, the number of variables might change)	.N/I.	.Yes.
Chaotic Continuous Stirred Tank Reactor (Wang *et al.* 2007)	Dynamic model of a Continuous Stirred Tank Reactor (Morningred *et al.* 1990). In Lightbody & Irwin (1997) it is described as a realistic non-linear case study. We cannot get the original paper to verify the claim.		.Yes. (HANDLED)	.No.	.Partly. (the future dynamic can be predicted, but the errors/ disturbances are not)	.Yes. (.WINDOW. - the algorithm would decide how the problem change in the future)	.Yes.	.S.	Spec Satisfaction (Maintain stability)	The dynamics of the system are non-linear. Depending on the previous solutions the future state of the system might be stable or unstable or chaotic. In certain situations due to the chaotic dynamic there are oscillations (although not cyclic)	.Tracking. (.LEARN. - at each receding horizon solutions from the past are used to predict the future behaviour. Here the sense of tracking is applied in a greater extent: the optimiser (parameter estimator) might not use solutions from the previous step, but the predictor needs to use past solution and hence the whole dynamic control system follows the tracking approach)	.Yes. (parameter of dynamic systems)	.No.	.No.	.N/I.	.Yes.

Table 5 Continuous non-metaheuristic references (cont.)

References	Origin of real-world data	Notes	Time-linkage	Solved by EA / meta-heuristics?	Predictable	Visible	Constr. probs?	S/M-obj	Optimisation goal	Types of dynamics	Restart/Track	Factors that change: Parameters of obj func	Factors that change: Domain range	Factors that change: Number of variables	Factors that change: Other prm	Constraints
Zymomonas Mobilis Reactor (Wang *et al.* 2007)	The dynamic model is simulated from a real reactor system described in (Daugulis *et al.* 1997)		.Yes. (HANDLED)	.No.	.Partly. (the future dynamic can be predicted, but the errors/ disturbances are not)	.Yes. (.WINDOW. - the optimisation process is divided into time-windows and hence changes are assumed to occur at the beginning of each time window)	.Yes.	.S.	Spec Satisfaction (Maintain stability)	The dynamics of the system are non-linear. Depending on the previous solutions the future state of the system might be stable or unstable or chaotic. In certain situations due to the chaotic dynamic there are oscillations (although not cyclic)	.Tracking. (.LEARN. - at each receding horizon solutions from the past are used to predict the future behaviour)	.Yes. (parameter of dynamic systems)	.No.	.No.	.N/I.	.Yes.

..

Bibliography

Abbass H A & Deb K (2003). Searching under Multi-evolutionary Pressures., *in Proceedings of the Evolutionary Multi-Criterion Optimization, Second International Conference, EMO 2003*, pp. 391–404.

Ahmad A Z & Liu K Z (2008). A new model predictive control approach to DC-DC converters based on combinatory optimization, *in Proceedings - 34th Annual Conference of the IEEE Industrial Electronics Society, IECON 2008*, Orlando, FL, United states, pp. 460 – 465.

Aickelin U & Dowsland K (2000). Exploiting Problem Structure in a Genetic Algorithm Approach to a Nurse Rostering Problem, *Journal of Scheduling* **3**, 139–153.

Akanle O & Zhang D (2008). Agent-based model for optimising supply-chain configurations, *International Journal of Production Economics* **115**(2), 444 – 60.

Alba E & Sarasola B (2010*a*). ABC, a New Performance Tool for Algorithms Solving Dynamic Optimization Problems, *in Proceedings of the 2010 IEEE World Congress on Computational Intelligence (WCCI'10)*, pp. 734–740.

Alba E & Sarasola B (2010*b*). Measuring Fitness Degradation in Dynamic Optimization Problems, *in Proceedings of the European Workshops on Applications of Evolutionary Computation, EvoApplicatons 2010, Part I*, pp. 572–581.

Alba E, Saucedo Badia J & Luque G (2007). A Study of Canonical GAs for NSOPs, *in* Doerner et al, ed., *Metaheuristics*, Vol. 39 of *Operations Research/Computer Science Interfaces Series*, Springer US, pp. 245–260.

Andersen H C (1991), An Investigation into Genetic Algorithms, and the Relationship between Speciation and the Tracking of Optima in Dynamic Functions, Honours thesis, Queensland University of Technology, Brisbane, Australia.

Andrews M & Tuson A L (2005). Dynamic Optimisation: A Practitioner Requirements Study, *in Proceedings of the The 24th Annual Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2005)*, London, UK.

Angeline P J (1997). Tracking extrema in dynamic environments, *in* P J Angeline, R G Reynolds, J R McDonnell & R Eberhart, eds, *Sixth International Conference on Evolutionary Programming*, Vol. 1213 of *LNCS*, Springer, pp. 335–345.

Aragon V S & Esquivel S C (2004). An evolutionary algorithm to track changes of optimum value locations in dynamic environments, *Journal of Computer Science and Technology* **4**(3), 127–134.

Araujo L & Merelo J J (2007). A genetic algorithm for dynamic modelling and prediction of activity in document streams, *in GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ACM, New York, NY, USA, pp. 1896–1903.

Arnold D V & Beyer H G (2002). Random Dynamics Optimum Tracking with Evolution Strategies, *in* J Merelo, P Adamidis, H G Beyer, J Fernández-Villacañas & H P Schwefel, eds, *Parallel Problem Solving from Nature*, Springer, Heidelberg, pp. 3–12.

Arnold D V & Beyer H G (2006). Optimum Tracking with Evolution Strategies, *Evolutionary Computation* **14**(3), 291–308.

Atkin J A D, Burke E K, Greenwood J S & Reeson D (2008). On-line decision support for take-off runway scheduling with uncertain taxi times at London Heathrow airport, *Journal of Scheduling* **11**(5), 323–346.

Ayvaz D, Topcuoglu H & Gurgen F (2006). A comparative study of evolutionary optimization techniques in dynamic environments, *in GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, ACM, pp. 1397–1398.

Bäck T (1998). On the Behavior of Evolutionary Algorithms in Dynamic Environments, *in IEEE International Conference on Evolutionary Computation*, IEEE, pp. 446–451.

Back T, Fogel D B & Michalewicz Z, eds (1997). *Handbook of Evolutionary Computation*, IOP Publishing Ltd., Bristol, UK, UK.

Bartusch M, Mohring R H & Radermacher F J (1988). Scheduling project networks with resource constraints and time windows, *Annals of Operations Research* **16**(1-4), 201–240.

Basu M (2005). A simulated annealing-based goal-attainment method for economic emission load dispatch of fixed head hydrothermal power systems, *International Journal of Electrical Power & Energy Systems* **27**(2), 147 – 153.

Beasley J E, Krishnamoorthy M, Sharaiha Y M & Abramson D (2004). Displacement problem and dynamically scheduling aircraft landings, *Journal of the Operational Research Society* **55**, 54–65.

Beccuti A, Mariéthoz S, Cliquennois S, Wang S & Morari M (2009). Explicit Model Predictive Control of DC-DC Switched Mode Power Supplies with Extended Kalman Filtering, *IEEE Transactions on Industrial Electronics* **56**(6), 1864 – 1874.

Bendtsen C N & Krink T (2002). Dynamic Memory Model for Non-Stationary Optimization, *in Congress on Evolutionary Computation*, IEEE, pp. 145–150.

Beyer H G & Sendhoff B (2007). Robust optimization - A comprehensive survey, *Computer Methods in Applied Mechanics and Engineering* **196**(33-34), 3190–3218.

Bianco L, DellOlmo P & Giordani S (2006). Scheduling models for air traffic control in terminal areas, *Journal of Scheduling* **9**(3), 223–253.

Bird S & Li X (2007). Informative performance metrics for dynamic optimisation problems, *in GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ACM, New York, NY, USA, pp. 18–25.

Blackwell T (2007). Particle Swarm Optimization in Dynamic Environment, *in* S Yang, Y S Ong & Y Jin, eds, *Evolutionary Computation in Dynamic and Uncertain Environments*, Studies in Computational Intelligence, Springer-Verlag, NJ, USA, pp. 28–49.

Blackwell T & Branke J (2006). Multiswarms, exclusion, and anti-convergence in dynamic environments., *IEEE Trans. Evolutionary Computation* **10**(4), 459–472.

Blackwell T M & Bentley P J (2002). Dynamic Search with Charged Swarms, *in* W B L et al., ed., *Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, pp. 19–26.

Bosman P A N (2005). Learning, Anticipation and Time-Deception in Evolutionary Online Dynamic Optimization, *in* S Yang & J Branke, eds, *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization*.

Bosman P A N (2007). Learning and Anticipation in Online Dynamic Optimization, *in* S Yang, Y S Ong & Y Jin, eds, *Evolutionary Computation in Dynamic and Uncertain Environments*, Vol. 51 of *Studies in Computational Intelligence*, Springer, pp. 129–152.

Bosman P A N & Poutré H L (2007). Learning and anticipation in online dynamic optimization with evolutionary algorithms: the stochastic case, *in GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, ACM, New York, NY, USA, pp. 1165–1172.

Branke J (1999). Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems, *in Congress on Evolutionary Computation CEC99*, Vol. 3, IEEE, pp. 1875–1882.

Branke J (2001*a*). Evolutionary Approaches to Dynamic Environments - updated survey, *in GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pp. 27–30.

Branke J (2001*b*). *Evolutionary Optimization in Dynamic Environments*, Kluwer.

Branke J (2003). Evolutionary Approaches to Dynamic Optimization Problems – Introduction and Recent Trends, *in* J Branke, ed., *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pp. 2–4.

Branke J, Kaußler T, Schmidt C & Schmeck H (2000). A multi-population approach to Dynamic Optimization Problems, *in Adaptive Computing in Design and Manufacturing 2000*, Springer.

Branke J & Mattfeld D (2005). Anticipation and flexibility in dynamic scheduling, *International Journal of Production Research* **43**(15), 3103–3129.

Branke J, Salihoglu E & Uyar S (2005). Towards an Analysis of Dynamic Environments, *in* H G Beyer & others, eds, *Genetic and Evolutionary Computation Conference*, ACM, pp. 1433–1439.

Branke J & Schmeck H (2003). Designing Evolutionary Algorithms for Dynamic Optimization Problems, *in* S Tsutsui & A Ghosh, eds, *Theory and Application of Evolutionary Computation: Recent Trends*, Springer, pp. 239–262.

Budge S, Ingolfsson A & Zerom D (2008). Empirical analysis of ambulance travel times: the case of Calgary Emergency Medical Services, Technical report, School of Business, University of Alberta, Canada. submitted to Management Science; manuscript no.MS-0001-1922.65. Accessed 05/06/2009.

Bui L, Abbass H & Branke J (2005). Multiobjective optimization for dynamic environments, *in Congress on Evolutionary Computation*, Vol. 3, IEEE press, pp. 2349 – 2356.

Bulitko V, Sturtevant N, Lu J & Yau T (2007). Graph abstraction in real-time heuristic search, *Journal of Artificial Intelligence Research* **30**(1), 51–100.

Carlisle A & Dozier G (2000). Adapting particle swarm optimisationto dynamic environments, *in Proceedings of the International Conference on Artificial Intelligence*, pp. 429–434.

Cedeno W & Vemuri V R (1997). On the Use of Niching for Dynamic Landscapes, *in International Conference on Evolutionary Computation*, IEEE.

Chaer R & Monzon P (2008). Stability conditions for a stochastic dynamic optimizer for optimal dispatch policies in power systems with hydroelectrical generation, *in IEEE/PES Transmission and Distribution Conference and Exposition: Latin America*, pp. 1–5.

Cheng H & Yang S (2010). Multi-population Genetic Algorithms with Immigrants Scheme for Dynamic Shortest Path Routing Problems in Mobile Ad Hoc Networks, *in* Di Chio et al, ed., *Applications of Evolutionary Computation*, Vol. 6024 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 562–571.

Cobb H G (1990). An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuouis, Time-Dependent Nonstationary Environments, Technical Report AIC-90-001, Naval Research Laboratory, Washington, USA.

Cobb H G & Grefenstette J J (1993). Genetic Algorithms for Tracking Changing Environments, *in International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 523–530.

Coello Coello C A (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art, *Computer Methods in Applied Mechanics and Engineering* **191**(11-12), 1245–1287.

Collingwood E, Corne D & Ross P (1996). Useful diversity via multiploidy, *in Proceedings of IEEE International Conference on Evolutionary Computation, 1996*, pp. 810 –813.

Daugulis A J, McLellan P J & Li J (1997). Experimental investigation and modeling of oscillatory behavior in the continuous culture of Zymomonas mobilis, *Biotechnology and Bioengineering* **56**(1), 99–105.

de França F O & Von Zuben F J (2009). A dynamic artificial immune algorithm applied to challenging benchmarking problems, *in Proceedings of the Eleventh IEEE Congress on Evolutionary Computation, CEC'09*, IEEE Press, Piscataway, NJ, USA, pp. 423–430.

de Prada C, Sarabia D, Cristea S & Mazaeda R (2008). Plant-wide Control of a Hybrid Process, *International Journal of Adaptive Control and Signal Processing* **22**(2), 124–141.

Deb K, Rao U B & Karthik S (2007). Dynamic Multi-objective Optimization and Decision-Making Using Modified NSGA-II: A Case Study on Hydro-thermal Power Scheduling, *in Evolutionary Multi-Criterion Optimization, 4th International Conference, EMO 2007, Matsushima, Japan, March 5-8, 2007, Proceedings*, Vol. 4403 of *Lecture Notes in Computer Science*, Springer, pp. 803–817.

del Real A J, Arce A & Bordons C (2007). Development and experimental validation of a PEM fuel cell dynamic model, *Journal of Power Sources* **173**(1), 310 – 324.

Dini D, van Lent M, Carpenter P & Iyer K (2006). Building robust planning and execution systems for virtual worlds, *in Proceedings of the Artificial Intelligence and Interactive Digital Entertainment converence (AIIDE)*, pp. 29–35.

Dowsland K A (1998). Nurse scheduling with tabu search and strategic oscillation, *European Journal of Operational Research* **106**(2-3), 393 – 407.

Droste S (2002). Analysis of the (1+1) EA for a dynamically changing OneMax-variant, *in Congress on Evolutionary Computation*, IEEE Press, pp. 55–60.

Droste S (2003). Analysis of the (1+1) EA for a dynamically bitwise changing OneMax, *in* E Cantu-Paz, ed., *Genetic and Evolutionary Computation Conference*, Vol. 2723 of *LNCS*, Springer, pp. 909–921.

Eggermont J, Lenaerts T, Poyhonen S & Termier A (2001). Raising the Dead; Extending Evolutionary Algorithms with a Case-based Memory, *in* J F Miller & others, eds, *Genetic Programming, Proceedings of EuroGP'2001*, Vol. 2038, Springer, pp. 280–290.

Eiben A E (2001). Springer-Verlag, London, UK, chapter Evolutionary algorithms and constraint satisfaction: definitions, survey, methodology, and research directions, pp. 13–30.

Farina M, Deb K & Amato P (2004). Dynamic multiobjective optimization problems: test cases, approximations, and applications, *IEEE Transactions on Evolutionary Computation* **8**(5), 425–442.

Farmani R & Wright J A (2003). Self-adaptive fitness formulation for constrained optimization, *IEEE Trans. Evolutionary Computation* **7**(5), 445–455.

Feng W, Brune T, Chan L, Chowdhury M, Kuek C & Li Y (1997). Benchmarks for testing evolutionary algorithms, Technical report, Center for System and Control, University of Glasgow.

Fernández J L & Arcos J L (2010). Adapting Particle Swarm Optimization in Dynamic and Noisy Environments, *in Proceedings of the 2010 IEEE Congress on Evolutionary Computation CEC'2010*, pp. 765–772.

Fiacchini M, Alamo T, Alvarado I & Camacho E F (2008). Safety Verification and Adaptive Model Predictive Control of the Hybrid Dynamics of a Fuel Cell System, *International Journal of Adaptive Control and Signal Processing* **22**(3), 142–160.

Floudas C, Pardalos P, Adjiman C, Esposito W, Gumus Z, Harding S, Klepeis J, Meyer C & Schweiger C (1999). *Handbook of Test Problems in Local and Global Optimization*, Vol. 33 of *Noncovex Optimization and Its Applications*, Kluwer Academic Publishers.

Fogel L, Owens A & Walsh M (1966). *Artificial intelligence through simulated evolution*, John Wiley & Sons Inc,.

Gao J & Sheng Z (2008). Research for dynamic vehicle routing problem with time windows in real city environment, *in Proceedings of the 2008 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, Vol. vol.2, Piscataway, NJ, USA, pp. 3052 – 6.

Gaspar A & Collard P (1999). From GAs to Artificial Immune Systems: Improving Adaptation in Time Dependent Optimization, *in Congress on Evolutionary Computation*, Vol. 3, IEEE, pp. 1859 – 1866.

Gleicher M & Ferrier N (2002). Evaluating Video-Based Motion Capture, *in CA '02: Proceedings of the Computer Animation*, IEEE Computer Society, Washington, DC, USA, pp. 75–80.

Goh C K & Tan K C (2009*a*). A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization, *IEEE Transactions on Evolutionary Computation* **13**(1), 103–127.

Goh C K & Tan K C (2009*b*). *Evolutionary Multi-objective Optimization in Uncertain Environments: Issues and Algorithms*, Springer Publishing Company, Incorporated.

Goldberg D E & Smith R E (1987). Nonstationary Function Optimization using Genetic Algorithms with Dominance and Diploidy, *in* J J Grefenstette, ed., *International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, pp. 59–68.

Graves S C & Willems S P (2005). Optimizing the Supply Chain Configuration for New Products, *Management Science* **51**(8), 1165–1180.

Grefenstette J J (1992). Genetic algorithms for changing environments, *in* R Maenner & B Manderick, eds, *Parallel Problem Solving from Nature 2*, North Holland, pp. 137–144.

Grefenstette J J (1999). Evolvability in Dynamic Fitness Landscapes: A Genetic Algorithm Approach, *in Congress on Evolutionary Computation*, Vol. 3, IEEE, pp. 2031–2038.

Grefenstette J J, Gopal R, Rosmaita B & van Gucht D (1985). Genetic algorithm for the TSP, *in Proceedings of the First International Conference on Genetic Algorithms*, pp. 160–168.

Guntsch M, Branke J, Middendorf M & Schmeck H (2000). AGO strategies for dynamic TSP, *in Proceedings of ANTS Workshop*, pp. 59–62.

Guntsch M & Middendorf M (2002). Applying population-based AGO to dynamic optimization problems, *in Lecture Notes in ComputerScience, vol.2463, Procedings of ANTS Workshop*, pp. 111–122.

Hadj-Alouane A B & Bean J C (1997). A Genetic Algorithm for the Multiple-Choice Integer Program, *Operations Research* **45**, 92–101.

Hamida S B & Petrowski A (2000). The Need for Improving the Exploration Operators for Constrained Optimization Problems, *in Proceedings of the Congress on Evolutionary Computation 2000, CEC'00*, Vol. 2, pp. 1176–1183.

Hamida S B & Schoenauer M (2002). ASCHEA: new results using adaptive segregational constraint handling, *in Proceedings of the 2002 Congress on Evolutionary Computation, 2002. CEC'02.*, Vol. 1, IEEE Press, Los Alamitos, CA, USA, pp. 884–889.

Hatzakis I & Wallace D (2006). Dynamic multi-objective optimization with evolutionary algorithms: a forward-looking approach, *in GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, ACM Press, New York, NY, USA, pp. 1201–1208.

Haugwitz S, Hagander P & Norén T (2007). Modeling and control of a novel heat exchange reactor, the Open Plate Reactor, *Control Engineering Practice* **15**(7), 779 – 792.

Houwing M, Negenborn R, Heijnen P, De Schutter B & Hellendoorn H (2007). Least-Cost Model Predictive Control of Residential Energy Resources when Applying μCHP, *in Proceedings of the Power Tech 2007 conference*, Lausanne, Switzerland. Paper 291.

Hu X & Eberhart R (2002). Adaptive particle swarm optimisation: detection and response to dynamic systems, *in Proceedings of the IEEE Congress on Evolutionary Computation, CEC2002*, pp. 1666–1670.

Huang S C & Wu T K (2008). Integrating GA-based time-scale feature extractions with SVMs for stock index forecasting, *Expert Systems with Applications* **35**(4), 2080 – 2088.

Ingolfsson A, Budge S & Erkut E (2008). Optimal ambulance location with random delays and travel times, *Health Care Management Science* **11**(3), 262–274.

Ioannou P, Chassiakos A, Jula H & Unglaub R (2002). Dynamic optimization of cargo movement by trucks in metropolitan areas with adjacent ports, Technical report, METRANS Transportation Center, University of Southern California, Los Angeles, CA 90089, USA. **URL:** *www.metrans.org/research/final/00-15_Final.htm*

Isaacs A, Puttige V R, Ray T, Smith W & Anavatti S G (2008). Development of a memetic algorithm for Dynamic Multi-Objective Optimization and its applications for online neural network modeling of UAVs., *in Proceedings of the International Joint Conference on Neural Networks, IJCNN 2008*, IEEE, pp. 548–554.

Janson S & Middendorf M (2005). A hierarchical particle swarm optimizer and its adaptive variant, *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics* **35**, 1272–1282.

Janson S & Middendorf M (2006). A hierarchical particle swarm optimizer for noisy and dynamic environments., *Genetic Programming and Evolvable Machines* **7**(4), 329–354.

Jatmiko W, Mursanto P, Kusumoputro B, Sekiyama K & Fukuda T (2008). Modified PSO algorithm based on flow of wind for odor source localization problems in dynamic environments, *WSEAS Transaction on Systems* **7**(2), 106–113.

Jatmiko W, Sekiyama K & Fukuda T (2006). A PSO-based Mobile Sensor Network for Odor Source Localization in Dynamic Environment: Theory, Simulation and Measurement, *in IEEE Congress on Evolutionary Computation, 2006. CEC 2006.*, pp. 1036 – 1043.

Jin N, Termansen M, Hubacek K, Holden J & Kirkby M (2007). Adaptive farming strategies for dynamic economic environment, *in Proceedings of the IEEE Congress on Evolutionary Computation CEC2007, 2007*, pp. 1213–1220.

Jin Y & Branke J (2005). Evolutionary Optimization in Uncertain Environments—A Survey, *IEEE Transactions on Evolutionary Computation* **9**(3), 303–317.

Jin Y, Oh S & Jeon M (2010). Incremental approximation of nonlinear constraint functions for evolutionary constrained optimization, *in Proceedings of the 2010 IEEE Congress on Evolutionary Computation, CEC'2010*, pp. 2966–2973.

Jin Y & Sendhoff B (2004). Constructing dynamic optimization test problems using the multi-objective optimization concept, *in* G R Raidl, ed., *Applications of evolutionary computing*, Vol. 3005 of *LNCS*, Springer, pp. 525–536.

Joines J & Houck C (1994). On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs, *in* D Fogel, ed., *Proceedings of the first IEEE Conference on Evolutionary Computation*, IEEE Press, pp. 579–584.

Kanoh H (2007). Dynamic route planning for car navigation systems using virus genetic algorithms, *International Journal of Knowledge-based and Intelligent Engineering Systems* **11**(1), 65–78.

Kanoh H & Hara K (2008). Hybrid genetic algorithm for dynamic multi-objective route planning with predicted traffic in a real-world road network, *in GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, ACM, New York, NY, USA, pp. 657–664.

Kashtan N, Noor E & Alon U (2007). Varying environments can speed up evolution, *Proceedings of the National Academy of Sciences* **104**(34), 13711–13716. in: "Population Structure and Artificial Evolution".

Kim H (2006). Target Exploration for Disconnected Feasible Regions in Enterprise-Driven Multilevel Product Design, *American Institute of Aeronautics and Astronautics Journal* **44**(1), 67–77.

Kiselev I & Alhajj R (2008). An adaptive multi-agent system for continuous learning of streaming data, *in Proceedings of the 2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, Vol. vol.2, Piscataway, NJ, USA, pp. 148 – 153.

Ko P C, Lin P C & Shih C S (2008). Stock valuation and dynamic asset allocation with genetic algorithm and cubic spline, *in Proceedings of the 2008 International Conference on Machine Learning and Cybernetics (ICMLC)*, Vol. vol.7, Piscataway, NJ, USA, pp. 3997 – 4000.

Lewis J, Hart E & Ritchie G (1998). A Comparison of Dominance Mechanisms and Simple Mutation on Non-stationary Problems, *in* A E Eiben, T Bäck, M Schoenauer & H P Schwefel, eds, *Parallel Problem Solving from Nature*, number 1498 *in* 'LNCS', Springer, pp. 139–148.

Li C & Yang S (2009). A clustering particle swarm optimizer for dynamic optimization, *in Proceedings of the Eleventh IEEE Congress on Evolutionary Computation, CEC'09*, IEEE Press, Piscataway, NJ, USA, pp. 439–446.

Li C, Yang S, Nguyen T T, Yu E L, Yao X, Jin Y, Beyer H G & Suganthan P N (2008). Benchmark Generator for CEC 2009 Competition on Dynamic Optimization, Technical report, University of Leicester and University of Birmingham, UK.

Li X, Branke J & Blackwell T (2006). Particle swarm with speciation and adaptation in a dynamic environment, *in GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, ACM Press, New York, NY, USA, pp. 51–58.

Li X, Branke J & Kirley M (2007). On performance metrics and particle swarm methods for dynamic multiobjective optimization problems, *in Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007*, pp. 576–583.

Liekens A, Eikelder H & Hilbers P (2003). Finite Population Models of Dynamic Optimization with Alternating Fitness Functions, *in* J Branke, ed., *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pp. 19–23.

Liekens A M L (2005), Evolution of Finite Populations in Dynamic Environments, PhD thesis, Technische Universität Eindhoven.

Lightbody G & Irwin G W (1997). Nonlinear Control Structures Based on Embedded Neural System Models, *IEEE Transactions on Neural Networks* **8**(3), 553–567.

Liu C A (2008*a*). New Dynamic Constrained Optimization PSO Algorithm, *in ICNC '08: Proceedings of the 2008 Fourth International Conference on Natural Computation*, IEEE Computer Society, pp. 650–653.

Liu L (2008*b*), Real-time contaminant source characterization in water distribution systems, PhD thesis, North Carolina State University.

Liu L, Zechman E M, Brill E D, Mahinthakumar G, Ranjithan S & Uber J (2006). Adaptive Contamination Source Identification in Water Distribution Systems Using an Evolutionary Algorithm-based Dynamic Optimization Procedure, *in Proceedings of the 8th Annual Water Distribution Systems Analysis Symposium*, Cincinnati, OH.

Liu T K, Liu Y T, Chen C H, Chou J H, Tsai J T & Ho W H (2007). Multi-objective optimization on robust airline schedule recover problem by using evolutionary computation, *in Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, 2007. ISIC.*, pp. 2396–2401.

Long C E, Polisetty P K & Gatzke E P (2007). Deterministic global optimization for nonlinear model predictive control of hybrid dynamic systems, *International Journal of Robust and Nonlinear Control* **17**(13), 1232 – 1250.

Louis S J & Xu Z (1996). Genetic Algorithms for Open Shop Scheduling and Re-Scheduling, *in* M E Cohen & D L Hudson, eds, *ISCA Eleventh International Conference on Computers and their Applications*, pp. 99–102.

Lung R I & Dumitrescu D (2007). A new collaborative evolutionary-swarm optimization technique, *in GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, ACM, New York, NY, USA, pp. 2817–2820.

Mariéthoz S, Wijekoon T & Wheeler P (2008). Analysis, Control and Comparison of Hybrid Two-stage Matrix Converters for Increased Voltage Transfer Ratio and Unity Power Factor, *IEEJ Transactions on Industry Applications* **128-D**(7), 892 – 900.

Martinez M, Moron A, Robledo F, Rodriguez-Bocca P, Cancela H & Rubino G (2008). A GRASP algorithm using RNN for solving dynamics in a P2P live video streaming network, *in Proceedings of the 2008 8th International Conference on Hybrid Intelligent Systems (HIS)*, Piscataway, NJ, USA, pp. 447 – 452.

Menchinelli P & Bemporad A (2008). Hybrid model predictive control of a solar air conditioning plant., *European Journal of Control* **14**(s6), 501–515.

Mendes R & Mohais A (2005). DynDE: a differential evolution for dynamic optimization problems, *in Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, IEEE, pp. 2808–2815.

Merino A, Mazaeda R, Alves R, Rueda A, Acebes L F & de Prada C (2006). Sugar Factory Simulator For Operators Training, *in Proceedings of the 7th Symposium on Advances in Control Education ACE2006, 2006*, Madrid.

Mertens K, Holvoet T & Berbers Y (2006). The DynCOAA algorithm for dynamic constraint optimization problems, *in AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, ACM, New York, NY, USA, pp. 1421–1423.

Mezura-Montes E & Coello C A C (2005). A simple multimembered evolution strategy to solve constrained optimization problems, *IEEE Trans. Evolutionary Computation* **9**(1), 1–17.

Mezura-Montes E, ed. (2009). *Constraint-Handling in Evolutionary Optimization*, Springer Publishing Company, Berlin.

Michalewicz Z (1995). A Survey of Constraint Handling Techniques in Evolutionary Computation Methods, *in Proceedings of the 4th Annual Conference on Evolutionary Programming*, MIT Press, pp. 135–155.

Michalewicz Z (1997). Oxford University Press, chapter Constraint-Handling Techniques : Decoders., pp. C5.3:1–C5.3:3.

Michalewicz Z (n.d.), 'The second version of Genocop III: a system which handles also nonlinear constraints'. [Accessed February 2009].
URL: *http://www.cs.adelaide.edu.au/ zbyszek/EvolSyst/gcopIII10.tar.Z*

Michalewicz Z & Nazhiyath G (1995). Genocop III: A co-evolutionary algorithm for numerical optimization with nonlinear constraints, *in* D B Fogel, ed., *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, IEEE Press, Piscataway, New Jersey, pp. 647–651.

Mills-Tettey G A, Stentz A & Dias M B (2008). Continuous-field path planning with constrained path-dependent state variables, *in ICRA 2008 Workshop on Path Planning on Costmaps.*

Mitra P & Venayagamoorthy G K (2008). Real time implementation of an artificial immune system based controller for a DSTATCOM in an electric ship power system, *in Conference Record - IAS Annual Meeting (IEEE Industry Applications Society)*, Edmonton, AB, Canada.

Morales K A & Quezada C (1998). A universal eclectic genetic algorithm for constrained optimization, *in Proceedings 6th European Congress on Intelligent & Soft Computing, EUFIT'98*, pp. 518–522.

Mori N, Imanishi S, Kita H & Nishikawa Y (1997). Adaptation to changing environments by Means of the Memory Based Thermodynamical Genetic Algorithm, *in* T Bäck, ed., *International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 299–306.

Mori N, Kita H & Nishikawa Y (1996). Adaptation to a Changing Environment by Means of the Thermodynamical Genetic Algorithm, *in* H M Voigt, ed., *Parallel Problem Solving from Nature*, number 1141 *in* 'LNCS', Springer Verlag Berlin, pp. 513–522.

Mori N, Kita H & Nishikawa Y (1998). Adaptation to a Changing Environment by Means of the Feedback Thermodynamical Genetic Algorithm, *in* A E Eiben, T Bäck, M Schoenauer & H P Schwefel, eds, *Parallel Problem Solving from Nature*, number 1498 *in* 'LNCS', Springer, pp. 149–158.

Morimoto T, Ouchi Y, Shimizu M & Baloch M (2007). Dynamic optimization of watering Satsuma mandarin using neural networks and genetic algorithms, *Agricultural Water Management* **93**(1-2), 1–10.

Morningred J, Paden B, Seborg D & Mellichamp D (1990). An adaptive non-linear predictive control, *in Proceedings of American control conference*, p. 1614Ű1619.

Morrison R (2003). Performance Measurement in Dynamic Environments, *in* J Branke, ed., *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pp. 5–8.

Morrison R & De Jong K (2002). Measurement of Population Diversity, *in* P Collet, C Fonlupt, J K Hao, E Lutton & M Schoenauer, eds, *Artificial Evolution*, Vol. 2310 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 1047–1074.

Morrison R W (2004). *Designing Evolutionary Algorithms for Dynamic Environments*, number ISBN 3-540-21231-0, Springer-Verlag, Berlin.

Morrison R W & DeJong K A (1999). A Test Problem Generator for Non-Stationary Environments, *in Congress on Evolutionary Computation*, Vol. 3, IEEE, pp. 2047–2053.

Moser I (2007). Review - All Currently Known Publications On Approaches Which Solve the Moving Peaks Problem, Technical report, Swinburne University of Technology, Melbourne, Australia.

Moser I & Hendtlass T (2007*a*). A simple and efficient multi-component algorithm for solving dynamic function optimisation problems, *in Proceedings of the IEEE Congress on Evolutionary Computation, 2007. CEC 2007.*, pp. 252–259.

Moser I & Hendtlass T (2007*b*). Solving Dynamic Single-Runway Aircraft Landing Problems With Extremal Optimisation, *in IEEE Symposium on Computational Intelligence in Scheduling*.

Ng K P & Wong K C (1995). A New Diploid Scheme and Dominance Change Mechanism for Non-Stationary Function Optimization, *in Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 159–166.

Ngo S H, Jiang X, Le V T & Horiguchi S (2006). Ant-based survivable routing in dynamic WDM networks with shared backup paths, *The Journal of Supercomputing* **36**(3), 297–307.

Nguyen T T (2007). Classifying and characterising dynamic optimisation problems - a literature review, Technical report, School of Computer Science, Univesity of Birmingham.
URL: *http://www.cs.bham.ac.uk/~txn/unpublished/reports/DOP_classification.pdf*

Nguyen T T (2008*a*). A proposed real-valued dynamic constrained benchmark set, Technical report, School of Computer Science, Univesity of Birmingham.
URL: *http://www.cs.bham.ac.uk/~txn/Papers/DCOP_benchmark.pdf*

Nguyen T T (2008*b*). Tracking Optima in Dynamic Environments Using Evolutionary Algorithms - RSMG report 5, Technical report, School of Computer Science, University of Birmingham. [online].
URL: *http://www.cs.bham.ac.uk/~txn/unpublished/reports/Report_5_Thanh.pdf*

Nguyen T T & Yao X (2009*a*). Benchmarking and Solving Dynamic Constrained Problems, *in Proceedings of the IEEE Congress on Evolutionary Computation CEC2009*, IEEE Press, pp. 690–697.

Nguyen T T & Yao X (2009*b*). Dynamic Time-linkage Problem Revisited, *in* M Giacobini, P Machado, A Brabazon, J McCormack & others, eds, *Proceedings of the 2009 European Workshops on Applications of Evolutionary Computation, EvoWorkshops 2009*, Vol. 5484 of *Lecture Notes in Computer Science*, pp. 735–744.

Nguyen T T & Yao X (2010*a*). Continuous Dynamic Constrained Optimisation - The Challenges, *IEEE Transactions on Evolutionary Computation (given the option of Revise for Acceptance. Revised version resubmitted.)* . [online].
URL: *http://www.cs.bham.ac.uk/~txn/Papers/Nguyen_Yao_DCOP.pdf*

Nguyen T T & Yao X (2010*b*). Solving dynamic constrained optimisation problems using repair methods, *IEEE Transactions on Evolutionary Computation (given the option of Revise for Acceptance)* .
URL: *http://www.cs.bham.ac.uk/~txn/Papers/Nguyen_Yao_dRepairGA.pdf*

Oppacher F & Wineberg M (1999). The Shifting Balance Genetic Algorithm: Improving the GA in a Dynamic Environment, *in* W Banzhalf, ed., *Genetic and Evolutionary Computation Conference*, Vol. 1, Morgan Kaufmann, pp. 504–510.

Orkin J (2006). 3 States & a Plan: The AI of F.E.A.R., *in Game Developer's Conference Proceedings*.

Orvosh D & Davis L (1993). Shall We Repair? Genetic Algorithms, Combinatorial Optimization and Feasibility Constraints, *in* S Forrest, ed., *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, Morgan Kauffman, p. 650.

Padula S, Gumbert C & Li W (2006). Aerospace applications of optimization under uncertainty, *Optimization and Engineering* **7**(3), 317–328.

Pantrigo J, Sanchez A, Montemayor A & Duarte A (2008). Multi-dimensional visual tracking using scatter search particle filter, *Pattern Recognition Letters* **29**(8), 1160 – 1174.

Parrott D & Li X (2006). Locating and tracking multiple dynamic optima by a particle swarm model using speciation., *IEEE Trans. Evolutionary Computation* **10**(4), 440–458.

Pina L & Botto M (2008). Dealing with uncertainty in the hybrid world, *in Proceedings of the Fifth International Conference on Informatics in Control, Automation and Robotics ICINCO 2008.*, Vol. vol.1, Madeira, Portugal.

Prata D M, Lima E L & Pinto J C (2006). Simultaneous Data Reconciliation and Parameter Estimation in Bulk Polypropylene Polymerizations in Real Time, *Macromolecular Symposia* **243**(1), 91–103.

Ramsey C L & Grefenstette J J (1993). Case-based initialization of genetic algorithms, *in* S Forrest, ed., *International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 84–91.

Rand W & Riolo R (2005*a*). Measurements for Understanding the Behavior of the Genetic Algorithm in Dynamic Environments: A Case Study using the Shaky Ladder Hyperplane-Defined Functions, *in* S Yang & J Branke, eds, *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization*.

Rand W & Riolo R (2005*b*). Shaky ladders, hyperplane-defined functions and genetic algorithms: systematic controlled observation in dynamic environments, *in* F Rothlauf & others, eds, *Applications of Evolutionary Computing*, Vol. 3449 of *LNCS*, Springer, pp. 600–609.

Richter H (2009). Detecting Change in Dynamic Fitness Landscapes, *in Proceedings of the IEEE 2009 Congress on Evolutionary Computation, CEC2009*, pp. 1613–1620.

Richter H (2010). Memory Design for Constrained Dynamic Optimization Problems, *in Proceedings of the European Conference on the Applications of Evolutionary Computation 2010, EvoApplications 2010*, Vol. 6024 of *Lecture Notes in Computer Science*, Springer, pp. 552–561.

Richter H & Yang S (2008). Memory Based on Abstraction for Dynamic Fitness Functions, *in* Mario Giacobini et al, ed., *Applications of Evolutionary Computing*, Vol. 4974 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 596–605.

Richter H & Yang S (2009). Learning behavior in abstract memory schemes for dynamic optimization problems, *Soft Computing* **13**(12), 1163–1173.

Riekert M, Malan K M & Engelbrect A P (2009). Adaptive genetic programming for dynamic classification problems, *in Proceedings of the Eleventh IEEE Congress on Evolutionary Computation, CEC'09*, IEEE Press, Piscataway, NJ, USA, pp. 674–681.

Rocha M, Neves J & Veloso A (2005). Evolutionary Algorithms for Static and Dynamic Optimization of Fed-batch Fermentation Processes, *in* B Ribeiro & others, eds, *Adaptive and Natural Computing Algorithms*, Springer, pp. 288–291.

Rohlfshagen P, Lehre P K & Yao X (2009). Dynamic evolutionary optimisation: An analysis of frequency and magnitude of change, *in Proceedings of the 2009 Genetic and Evolutionary Computation Conference GECCO'09*, pp. 1713–1720.

Rohlfshagen P & Yao X (2008). Attributes of Dynamic Combinatorial Optimisation, *in Proceedings of the 10th International Conference on Parallel Problem Solving From Nature PPSN'08*, Vol. 5361 of *Lecture Notes in Computer Science*, Springer, pp. 442–451.

Rohlfshagen P & Yao X (2010). On the Role of Modularity in Evolutionary Dynamic Optimisation, *in Proceedings of the 2010 IEEE Wolrd Congress on Computational Intelligence, WCCI 2010*, Spain, pp. 3539–3546.

Ronnewinkel C, Wilke C & Martinetz T (2000). Genetic Algorithms in Time-Dependent Environments, *in Theoretical Aspects of Evolutionary Computing*, Springer.

Rossi C, Abderrahim M & Díaz J C (2008). Tracking moving optima using kalman-based predictions, *Evolutionary Computation* **16**(1), 1–30.

Rowe J E (1999). Finding attractors for periodic fitness functions, *in* W Banzhaf & others, eds, *Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, pp. 557–563.

Rowe J E (2001). Cyclic attractors and quasispecies adaptability, *in* L Kallel, B Naudts & A Rogers, eds, *Theoretical Aspects of Evolutionary Computing*, Springer, pp. 251–259.

Rowe J E (2005). Population Dynamics of Genetic Algorithms, *in* L Bull & T Kovacs, eds, *Foundations of Learning Classifier Systems*, Vol. 183 of *Studies in Fuzziness and Soft Computing*, Springer Berlin / Heidelberg, pp. 19–43.

Runarsson T P & Yao X (2000). Stochastic ranking for constrained evolutionary optimization, *IEEE Trans. Evolutionary Computation* **4**(3), 284–294.

Ryan C (1996). The Degree of Oneness, *in First Online Workshop on Soft Computing*, pp. 43–49.

Salcedo-Sanz S (2009). A survey of repair methods used as constraint handling techniques in evolutionary algorithms, *Computer Science Review* **3**(3), 175 – 192.

Salomon R (1996). Re-evaluating Genetic Algorithm Performance Under Coordinate Rotation of Benchmark Functions: A Survey of Some Theoretical and Practical Aspects of Genetic Algorithms, *BioSystems* **39**, 263–278.

Salomon R & Eggenberger P (1997). Adaptation on the Evolutionary Time Scale: A Working Hypothesis and Basic Experiments, *in* J K Hao, E Lutton, E Ronald, M Schoenauer & D Snyers, eds, *3rd European Conference on Artificial Evolution*, number 1363 *in* 'LNCS', Springer, pp. 251–262.

Sane H & Guay M (2008). Minmax dynamic optimization over a finite-time horizon for building demand control, *in Proceedings of the American Control Conference*, Seattle, WA, United states, pp. 1469 – 1474.

Schlegel M & Marquardt W (2006). Adaptive switching structure detection for the solution of Dynamic Optimization Problems, *Industrial & engineering chemistry research* **45**(24), 8083–8094.

Simões A & Costa E (2003). An immune system-based genetic algorithm to deal with dynamic environments: Diversity and memory, *in* D W Pearson, N C Steele & R Albrecht, eds, *Proceedings of the Sixth international conference on neural networks and genetic algorithms (ICANNGA03)*, Springer, pp. 168–174.

Simões A & Costa E (2007). Improving memory's usage in evolutionary algorithms for changing environments, *in Proceedings of the 2007 IEEE Congress on Evolutionary Computation, CEC'07*, pp. 276–283.

Simões A & Costa E (2009). Improving prediction in evolutionary algorithms for dynamic environments, *in* Raidl et al, ed., *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, ACM, Montreal, Québec, Canada, pp. 875–882.

Simões A & Costa E (2008). Evolutionary Algorithms for Dynamic Environments: Prediction Using Linear Regression and Markov Chains, *in* G Rudolph, T Jansen, S Lucas, C Poloni & N Beume, eds, *Parallel Problem Solving from Nature Ű PPSN X*, Vol. 5199 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 306–315.

Singh H K, Isaacs A, Nguyen T T, Ray T & Yao X (2009). Performance of Infeasibility Driven Evolutionary Algorithm (IDEA) on Constrained Dynamic Single Objective Optimization Problems, *in Proceedings of the IEEE Congress on Evolutionary Computation CEC2009*, IEEE Press, Trondheim, Norway„ pp. 3127–3134.

Smith A E & Coit D W (1997). Constraint Handling Techniques—Penalty Functions, *in* T Bäck, D B Fogel & Z Michalewicz, eds, *Handbook of Evolutionary Computation*, Oxford University Press and Institute of Physics Publishing, chapter C5.2.

Soga T, Nanri T, Kurokawa M & Murakami K (2008). Effect of reordering internal messages in MPI broadcast according to the load imbalance, *in Proceedings of the Innovative Architecture for Future Generation High-Performance Processors and Systems*, Hilo, HI, United states, pp. 11 – 16.

Solomon M M (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints, *Operations Research* **35**(2), 254–265.

Sonntag C, Su W, Stursberg O & Engell S (2008). Optimized start-up control of an industrial-scale evaporation system with hybrid dynamics, *Control Engineering Practice* **16**(8), 976 – 990.

Srinivasarao M, Patwardhan S C & Gudi R D (2007). Nonlinear predictive control of irregularly sampled multirate systems using blackbox observers, *Journal of Process Control* **17**(1), 17 – 35.

Stanhope S A & Daida J M (1998). Optimal Mutation and Crossover Rates for a Genetic Algorithm Operating in a Dynamic Environment, *in Evolutionary Programming VII*, number 1447 *in* 'LNCS', Springer, pp. 693–702.

Stanhope S A & Daida J M (1999). Genetic Algorithm Fitness Dynamics in a Changing Environment, *in Congress on Evolutionary Computation*, Vol. 3, IEEE, pp. 1851–1858.

Suganthan P N, Hansen N, Liang J J, Deb K, Chen Y P, Auger A & Tiwari S (2005). Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization, Technical report, Nanyang Technology University, Singapore.

Summers S & Bewley T (2007). MPDopt: A versatile toolbox for adjoint-based model predictive control of smooth and switched nonlinear dynamic systems, *in Proceedings of the 46th IEEE Conference on Decision and Control 2007*, pp. 4785 – 4790.

Syswerda G (1991). Schedule optimization using genetic algorithms, *in* L Davis, ed., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, pp. 332–349.

Tafazoli S & Sun X (2006). Hybrid System State Tracking and Fault Detection Using Particle Filters, *IEEE Transactions on Control Systems Technology* **14**(6), 1078 –1087.

Takagi R, Kokago R, Goodman C & Weston P (2008). Optimisation of train re-scheduling using a genetic algorithm applied to the mimic panel state model of the overall rail network, *in Proceedings of the International Conference on Railway Engineering 2008. ICRE 2008*, Hong Kong, China, pp. 225 – 229.

Tawdross P, Lakshmanan S K & Konig A (2006). Intrinsic Evolution of Predictable Behavior Evolvable Hardware in Dynamic Environment, *in HIS '06: Proceedings of the Sixth International Conference on Hybrid Intelligent Systems*, IEEE Computer Society, p. 60.

Tfaili W, Dréo J & Siarry P (2007). Fitting of an ant colony approach to dynamic optimization through a new set of test functions, *International Journal of Computational Intelligence Research* **3**, 205–218.

Thompson J & Dowsland K A (1996). General Cooling Schedules for a Simulated Annealing Based Timetabling System, *in Selected papers from the First International Conference on Practice and Theory of Automated Timetabling*, Springer-Verlag, London, UK, pp. 345–363.

Thompson J M & Dowsland K A (1998). A robust simulated annealing based examination timetabling system, *Computers and Operations Research* **25**(7-8), 637–648.

Thornhill N F, Patwardhan S C & Shah S L (2008). A continuous stirred tank heater simulation model with applications, *Journal of Process Control* **18**(3-4), 347 – 360.

Tinos R & Yang S (2007). Continuous dynamic problem generators for evolutionary algorithms, *in Proceedings of the 2007 IEEE Congress on Evolutionary Computation*, pp. 236–243.

Tinos R & Yang S (2010). An Analysis of the XOR Dynamic Problem Generator Based on the Dynamical System, *in Proceedings of the Parallel Problems Solving from Nature (PPSN XI)*.

Toffolo A & Benini E (2003). Genetic diversity as an objective in multi-objective evolutionary algorithms, *Evolutionary Computation* **11**(2), 151–167.

Trojanowski K & Michalewicz Z (1999). Searching for Optima in Non-stationary Environments, *in Congress on Evolutionary Computation*, Vol. 3, IEEE, pp. 1843–1850.

Tsutsui S, Fujimoto Y & Ghosh A (1997). Forking Genetic Algorithms: GAs with Search Space Division Schemes., *Evolutionary Computation* **5**(1), 61–80.

Ursem R K (2000). Multinational GA Optimization Techniques in Dynamic Environments, *in* D Whitley, D Goldberg, E Cantu-Paz, L Spector, I Parmee & H G Beyer, eds, *Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, pp. 19–26.

Ursem R K, Krink T, Jensen M T & Michalewicz Z (2002). Analysis and modeling of control tasks in dynamic systems, *IEEE Transactions on Evolutionary Computation* **6**(4), 378–389.

Uyar A S & Harmanci A E (2005). A new population based adaptive domination change mechanism for diploid genetic algorithms in dynamic environments, *Soft Computing - A Fusion of Foundations, Methodologies and Applications* **9**(11), 803–814.

Van Veldhuizen D A (1999), Multiobjective evolutionary algorithms: classifications, analyses, and new innovations, PhD thesis, Air Force Institute of Technolog, Wright Patterson AFB, OH, USA. Adviser-Lamont, Gary B.

Vavak F, Fogarty T C & Cheng P (1995). Load Balancing Application of the Genetic Algorithm in a Nonstationary Environment, *in* T Fogarty, ed., *AISB Workshop on Evolutionary Computing*, number 993 *in* 'LNCS', Springer, pp. 224–233.

Vavak F, Fogarty T C & Jukes K (1996). A Genetic Algorithm with Variable Range of Local Search for Tracking Changing Environments, *in* H M Voigt, ed., *Parallel Problem Solving from Nature*, number 1141 *in* 'LNCS', Springer Verlag Berlin.

Vavak F, Jukes K A & Fogarty T C (1998). Performance of a Genetic Algorithm with Variable Local Search Range Relative to Frequency for the Environmental Changes, *in* K et al., ed., *International Conference on Genetic Programming*, Morgan Kaufmann.

Vavak F, Jukes K & Fogarty T C (1997*a*). Adaptive Combustion Balancing in Multiple Burner Boiler Using a Genetic Algorithm with Variable Range of Local Search, *in* T Bäck, ed., *International Conference on Genetic Algorithms*, Morgan Kaufmann, pp. 719–726.

Vavak F, Jukes K & Fogarty T C (1997*b*). Learning the Local Search Range for Genetic Optimisation in Nonstationary Environments, *in IEEE Intl. Conf. on Evolutionary Computation ICEC'97*, IEEE Publishing, pp. 355–360.

Velenis E & Tsiotras P (2008). Minimum-time travel for a vehicle with acceleration limits: theoretical analysis and receding-horizon implementation, *Journal of Optimization Theory and Applications* **138**(2), 275 – 296.

Venkatraman S & Yen G G (2005). A Generic Framework for Constrained Optimization Using Genetic Algorithms, *IEEE Trans. Evolutionary Computation* **9**(4), 424–435.

Vidotto A, Brown K N & Beck J C (2007). Managing restaurant tables using constraints, *Knowledge-Based Systems* **20**(2), 160 – 169.

Wang B, Wu W & Liu Y (2008). Dynamic channel assignment in wireless LANs, *in 2008 Workshop on Power Electronics and Intelligent Transportation System*, Piscataway, NJ, USA, pp. 12 – 17.

Wang F, Bahri P, Lee P & Cameron I (2007). A multiple model, state feedback strategy for robust control of non-linear processes, *Computers and Chemical Engineering* **31**(5-6), 410–418.

Wang J, Tao X & Cho H (2008). Microassembly of micro peg and hole using an optimal visual proportional differential controller, *Proceedings of the Institution of Mechanical Engineers, Part B (Journal of Engineering Manufacture)* **222**(B9), 1171 – 80.

Wang N, Ho K H & Pavlou G (2008). Adaptive multi-topology IGP based traffic engineering with near-optimal network performance, *in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 4982 LNCS, Singapore, pp. 654 – 666.

Wang Y & Wineberg M (2006). Estimation of evolvability genetic algorithm and dynamic environments, *Genetic Programming and Evolvable Machines* **7**(4), 355–382.

Weicker K (2000). An Analysis of Dynamic Severity and Population Size, *in* M Schoenauer, K Deb, G Rudolph, X Yao, E Lutton, J J Merelo & H P Schwefel, eds, *Parallel Problem Solving from Nature (PPSN VI)*, Vol. 1917 of *LNCS*, Springer.

Weicker K (2002). Performance Measures for Dynamic Environments, *in* J Merelo, P Adamidis, H G Beyer, J Fernández-Villacañas & H P Schwefel, eds, *Parallel Problem Solving from Nature*, Vol. 2439 of *LNCS*, Springer, pp. 64–73.

Weicker K (2003). *Evolutionary algorithms and dynamic optimization problems*, Der Andere Verlag.

Weicker K & Weicker N (1999). On Evolution Strategy Optimization in Dynamic Environments, *in Congress on Evolutionary Computation*, Vol. 3, pp. 2039–2046.

Weicker K & Weicker N (2000). Dynamic rotation and partial visibility, *in Congress on Evolutionary Computation*, pp. 1125–1131.

Wilkins D E, Smith S F, Kramer L A, Lee T J & Rauenbusch T W (2008). Airlift mission monitoring and dynamic rescheduling, *Engineering Applications of Artificial Intelligence* **21**(2), 141 – 155.

Woldesenbet Y G & Yen G G (2009). Dynamic evolutionary algorithm with variable relocation, *IEEE Transactions on Evolutionary Computation* **13**(3), 500–513.

Wolpert D H & Macready W G (1997). No Free Lunch Theorems for Optimization, *IEEE Transactions on Evolutionary Computation* **1**(1), 67–82.

Xiong Z (2008). A modified adaptive genetic BP neural network with application to financial distress analysis, *in Proceedings of the 2008 Second International Conference on Genetic and Evolutionary Computing (WGEC)*, Piscataway, NJ, USA, pp. 149 – 152.

Yang S (2004). Constructing dynamic test environments for genetic algorithms based on problem difficulty, *in Congress on Evolutionary Computation*, Vol. 2, pp. 1262–1269.

Yang S (2005*a*). Memory-Based Immigrants for Genetic Algorithms in Dynamic Environments, *in* H G Beyer & others, eds, *Genetic and Evolutionary Computation Conference*, ACM, pp. 1115–1122.

Yang S (2005*b*). Memory-enhanced univariate marginal distribution algorithms for dynamic optimization problems, *in Congress on Evolutionary Computation*, Vol. 3, IEEE press, pp. 2560–2567.

Yang S (2006*a*). Associative Memory Scheme for Genetic Algorithms in Dynamic Environments, *in* F Rothlauf & others, eds, *Applications of Evolutionary Computing*, Vol. 3907 of *LNCS*, Springer, pp. 788–799.

Yang S (2006*b*). A comparative study of immune system based genetic algorithms in dynamic environments, *in GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, ACM Press, New York, NY, USA, pp. 1377–1384.

Yang S (2006*c*). On the Design of Diploid Genetic Algorithms for Problem Optimization in Dynamic Environments, *in Proceedings of the 2006 IEEE Congress on Evolutionary Computation. CEC 2006.*, pp. 1362 –1369.

Yang S (2008). Genetic algorithms with memory- and elitism-based immigrants in dynamic environments, *Evolutionary Computation* **16**(3), 385–416.

Yang S & Richter H (2009). Hyper-learning for population-based incremental learning in dynamic environments, *in Proceedings of the Eleventh IEEE Congress on Evolutionary Computation, CEC'09*, IEEE Press, Piscataway, NJ, USA, pp. 682–689.

Yang S & Yao X (2003). Dual Population-Based Incremental Learning for Problem Optimization in Dynamic Environments, *in Proceedings of the 7th Asia Pacific Symposium on Intelligent and Evolutionary Systems*, pp. 49–56.

Yang S & Yao X (2005). Experimental study on population-based incremental learning algorithms for dynamic optimization problems, *Soft Computing - A Fusion of Foundations, Methodologies and Applications* **9**(11), 815–834.

Yang S & Yao X (2008). Population-Based Incremental Learning With Associative Memory for Dynamic Environments, *Evolutionary Computation, IEEE Transactions on* **12**(5), 542 –561.

Younes A (2006), Adapting Evolutionary Approaches for Optimization in Dynamic Environments, Doctor of philosophy (phd) in systems design engineering, Faculty of Engineering, University of Waterloo, Canada, Faculty of Engineering, University of Waterloo, Canada.

Yu D L, Yu D W & Gomm J (2006). Neural model adaptation and predictive control of a chemical process rig, *IEEE Transactions on Control Systems Technology* **14**(5), 828 –840.

Yu E L & Suganthan P N (2009). Evolutionary programming with ensemble of explicit memories for dynamic optimization, *in Proceedings of the Eleventh IEEE Congress on Evolutionary Computation, CEC'09*, IEEE Press, Piscataway, NJ, USA, pp. 431–438.

Yu X, Jin Y, Tang K & Yao X (2010). Robust Optimization over Time – A New Perspective on Dynamic Optimization Problems, *in Proceedings of the 2010 IEEE Wolrd Congress on Computational Intelligence, WCCI 2010*, Spain, pp. 3998–4003.

Zeng S, Chen G, Zheng L, Shi H, de Garis H, Ding L & Kang L (2006). A Dynamic Multi-Objective Evolutionary Algorithm Based on an Orthogonal Design, *in Proceedings of the IEEE Congress on Evolutionary Computation 2006 (CEC'06)*, IEEE Press, Vancouver, Canada, pp. 573–580.

Zeng S, Shi H, Kang L & Ding L (2007). Orthogonal Dynamic Hill Climbing Algorithm: ODHC, *in* S Yang, Y S Ong & Y Jin, eds, *Evolutionary Computation in Dynamic and Uncertain Environments*, Studies in Computational Intelligence, Springer-Verlag New York, Inc., pp. 79–105.

Zhang H & Li H (2007). A General Control Horizon Extension Method for Nonlinear Model Predictive Control, *Industrial & Engineering Chemistry Research* **46**(26), 9179–9189.

Zhou A, Jin Y, Zhang Q, Sendhoff B & Tsang E (2007). Prediction-Based Population Reinitialization for Evolutionary Dynamic Multi-objective Optimization, *in* S Obayashi, K Deb, C Poloni, T Hiroyasu & T Murata, eds, *Evolutionary Multi-Criterion Optimization*, Vol. 4403 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 832–846.